

METHOD AND DEVICE FOR DYNAMIC GENERATION MANAGEMENT OF COMPUTER MEMORY

Publication number: JP2000047931 (A)

Publication date: 2000-02-18

Inventor(s): GRARUP STEFFEN; BAK LARS

Applicant(s): SUN MICROSYSTEMS INC

Classification:


- international: G06F12/00; G06F9/44; G06F12/02; G06F12/00; G06F9/44; G06F12/02; (IPC1-7): G06F12/00; G06F9/44

- **European:** G06F12/02D2G4G

Application number: JP19990100480 19990302

Priority number(s): US19980077069P 19980306

Also published as:

 EP0940755 (A1)

EP0940755 (B1)

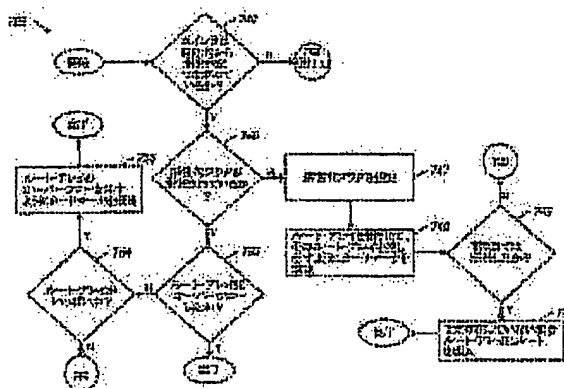
DE69932874 (T2)

 CN1236921 (A)

 CN100343821 (C)

Abstract of JP 2000047931 (A)

PROBLEM TO BE SOLVED: To execute a generation garbage collection in a computer memory by generating a new route array through the use of a prescribed marker when reference to a second object is stored after the execution of the second garbage collection. **SOLUTION:** When it is judged that a new generation pointer is not set (S746), the new generation pointer indicating that a pointer is connected from an old generation to the new one, is set (747). Then, the route array is assigned and a card mark is set so as to indicate the assigned route array (S748). Then, whether route array assignment has succeeded or not is judged (S749). When the assignment of the route array has succeeded, a route tracked by the route array is inserted (S750). In the meantime, at the time of failure in assignment, the card mark indicating that the route array overflows is set (S755).



Data supplied from the **esp@cenet** database — Worldwide

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2000-47931
(P2000-47931A)

(43) 公開日 平成12年2月18日 (2000.2.18)

(51) IntCl. ⁷	識別記号	F I	テーマコード* (参考)
G 0 6 F 12/00	5 9 1	G 0 6 F 12/00	5 9 1
9/44	5 3 0	9/44	5 3 0 S

審査請求 未請求 請求項の数25 O L 外国語出願 (全 78 頁)

(21) 出願番号 特願平11-100480

(22) 出願日 平成11年3月2日 (1999.3.2)

(31) 優先権主張番号 60/077, 069

(32) 優先日 平成10年3月6日 (1998.3.6)

(33) 優先権主張国 米国 (US)

(71) 出願人 591064003

サン・マイクロシステムズ・インコーポ
レーテッド

SUN MICROSYSTEMS, IN
CORPORATED

アメリカ合衆国 94303 カリフォルニア
州・パロ アルト・サン アントニオ ロ
ード・901

(72) 発明者 ステファン・グララップ

アメリカ合衆国 カリフォルニア州94303
パロ・アルト, クララ・ドライブ, 825

(74) 代理人 100096817

弁理士 五十嵐 孝雄 (外2名)

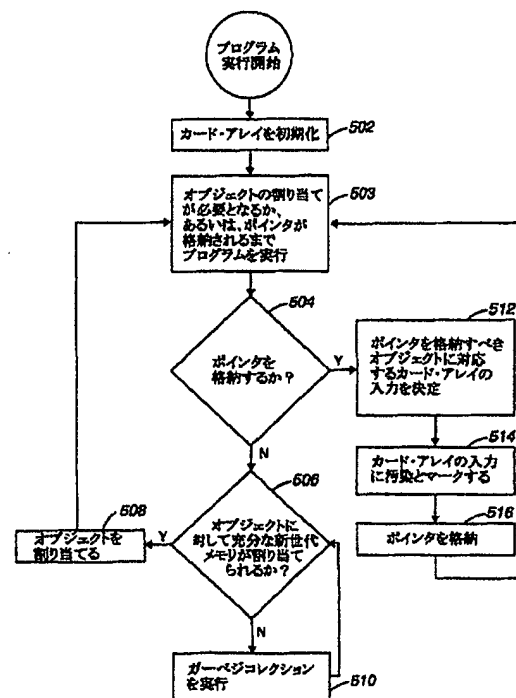
最終頁に続く

(54) 【発明の名称】 コンピュータメモリの世代動的管理のための方法及び装置

(57) 【要約】 (修正有)

【課題】 コンピュータメモリ内で世代ガーベジコレク
ションを実行するための方法を提案する。

【解決手段】 第1メモリ部と、関連マーカーを各々有
する複数のブロックに分割される第2メモリ部とを含む
メモリの第1メモリ部上で第1のガーベジコレクション
を実行し、また、第2メモリ部の複数のブロックのうち
の所定のブロック上で、第2のガーベジコレクションを
実行する。さらに、第2メモリ部の所定のブロック上
で、第3のガーベジコレクションを実行する。それ
には、所定のブロックに関連するマーカーによって表わ
されるステータスの少なくとも一部に基づいて、所定のブ
ロックがその所定のブロック内に含まれていない第2の
オブジェクトを参照するところの第1のオブジェクトを
含むか否かの判定を含む。



【特許請求の範囲】

【請求項 1】 コンピュータシステムに関係し、第 1 メモリ部と、関連マーカーを各々有する複数のブロックに分割される第 2 メモリ部と、を含むメモリを、動的に管理するためのコンピュータ実現方法であって、前記第 1 メモリ部上で、第 1 のガベージコレクションを実行する工程と、

前記第 2 メモリ部の前記複数のブロックのうちの所定のブロック上で、第 2 のガベージコレクションを実行する工程と、

前記第 2 メモリ部の前記所定のブロック上で、第 3 のガベージコレクションを実行する工程であって、前記第 3 のガベージコレクションは、前記所定のブロックに関連するマーカーによって表わされるステータスの少なくとも一部に基づいて、前記所定のブロックが、該所定のブロック内に含まれていない第 2 のオブジェクトを参照するところの第 1 のオブジェクトを含むか否かの判定を含み、前記ステータスは、前記第 2 のガベージコレクションの実行後に前記第 2 のオブジェクトへのリファレンスが記憶されたか否かの表示を含む工程と、

前記第 2 のガベージコレクションの実行後に前記第 2 のオブジェクトへのリファレンスが記憶された場合に、前記所定のマーカーを用いて、新たなルート・アレイを生成する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項 2】 請求項 1 に記載のコンピュータ実現方法であって、

前記第 1 のオブジェクトが前記所定のブロック内に含まれていない第 2 のオブジェクトを参照しないと判定された場合に、前記所定のブロックを解放する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項 3】 請求項 1 および請求項 2 のいずれか一方に記載のコンピュータ実現方法であって、

前記所定のブロックが第 2 のオブジェクトを参照する第 1 のオブジェクトを含むと判定された場合に、前記第 3 のガベージコレクションを実行する工程は、更に、前記所定のブロックマーカーを用いて、前記第 2 のオブジェクトが前記第 1 メモリ部内に存在するか否かを判定する工程と、

前記第 2 のオブジェクトが前記第 1 メモリ部内に存在すると判定された場合に、前記所定のブロックマーカーに関連するルート・アレイが前記第 2 のオブジェクトへのポインタを含むように、前記所定のブロックマーカーに関連する前記ルート・アレイを更新する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項 4】 請求項 1 および請求項 2 のいずれか一方に記載のコンピュータ実現方法であって、

前記所定のブロックが第 2 のオブジェクトを参照する第 1 のオブジェクトを含むと判定された場合に、前記第 3 のガベージコレクションを実行する工程は、更に、

前記所定のブロックマーカーを用いて、前記第 2 のオブジェクトが前記第 2 メモリ部内に存在するか否かを判定する工程と、

前記第 2 のオブジェクトが前記第 2 メモリ部内に存在すると判定された場合に、他のオブジェクトへのリファレンスを認定するように、前記所定のブロックマーカーに関連するルート・アレイを走査する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項 5】 請求項 4 に記載のコンピュータ実現方法であって、

他のオブジェクトへのリファレンスを追跡する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項 6】 請求項 1 および請求項 2 のいずれか一方に記載のコンピュータ実現方法であって、

前記複数のブロックが複数のトレインとして用意され、前記所定のブロックが、前記複数のトレインのうちの第 1 トレインに含まれると共に、前記第 3 のガベージコレクションを実行する工程が、更に、

前記ブロックに関連する前記マーカーによって表わされる前記ステータスの少なくとも一部に基づいて、前記第 2 のオブジェクトが、前記複数のトレインのうちの第 2 トレインに存在するか否かを判定する工程を備えることを特徴とするコンピュータ実現方法。

【請求項 7】 請求項 6 に記載のコンピュータ実現方法であって、

前記第 2 のオブジェクトが前記複数のトレインのうちの第 2 トレインに存在しないと判定された場合に、前記第 2 トレインを解放する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項 8】 請求項 1 および請求項 2 のいずれか一方に記載のコンピュータ実現方法であって、

前記複数のブロックが複数のカーとして用意され、前記複数のカーが、更に、複数のトレインとして用意され、前記所定のブロックが、前記複数のトレインのうちの所定のトレインを形成する複数のカーのうちの第 1 カーに含まれると共に、前記第 3 のガベージコレクションを実行する工程が、更に、

前記ブロックに関連する前記マーカーによって表わされる前記ステータスの少なくとも一部に基づいて、前記第 2 のオブジェクトが、前記所定のトレインを形成する前記複数のカーのうちの第 2 カーに存在するか否かを判定する工程を備えることを特徴とするコンピュータ実現方法。

【請求項 9】 請求項 8 に記載のコンピュータ実現方法であって、

前記第 2 のオブジェクトが前記所定のトレインの前記第 2 カーに存在しないと判定された場合に、前記第 2 カーを解放する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項 10】 コンピュータシステムに関係し、関連

マーカーを各々有する複数のブロックに分割されるメモリを、動的に管理するためにコンピュータ実現方法であって、

前記複数のブロックのうちの所定のブロック上で、第1のガーベジコレクションを実行する工程と、
前記所定のブロック上で、第2のガーベジコレクションを実行する工程であって、前記第2のガーベジコレクションは、前記所定のブロックに関連するマーカーによって表わされるステータスの少なくとも一部に基づいて、前記所定のブロックが、前記所定のブロック内に含まれていない第2のオブジェクトを参照するところの第1のオブジェクトを含むか否かの判定を含み、前記マーカーで表わされる前記ステータスは、前記第1のガーベジコレクションの実行後に前記第2のオブジェクトへのリファレンスが記憶されたか否かの表示を含む工程と、
前記第1のガーベジコレクションの実行後に前記第2のオブジェクトへのリファレンスが記憶された場合に、前記所定のブロックを用いて、新たなルート・アレイを生成する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項11】 請求項10に記載のコンピュータ実現方法であって、

前記第1のオブジェクトが前記所定のブロック内に含まれていない第2のオブジェクトを参照しないと判定された場合に、前記所定のブロックを解放する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項12】 請求項10および請求項11のいずれか一方に記載のコンピュータ実現方法であって、
前記複数のブロックが複数のトレインとして用意され、前記所定のブロックが、前記複数のトレインのうちの第1トレインに含まれると共に、前記ガーベジコレクションを実行する工程が、更に、
前記ブロックに関連する前記マーカーによって表わされる前記ステータスの少なくとも一部に基づいて、前記第2のオブジェクトが、前記複数のトレインのうちの第2トレインに存在するか否かを判定する工程を備え、
前記方法は、前記第2のオブジェクトが前記複数のトレインのうちの第2トレインに存在しないと判断された場合、前記第2トレインを解放する工程を更に備えることを特徴とするコンピュータ実現方法。

【請求項13】 請求項10に記載のコンピュータ実現方法であって、

追加メモリ部が設けられ、前記第2のオブジェクトが前記所定のブロック内に含まれないと判定された場合に、前記所定のブロック上で前記ガーベジコレクションを実行する工程が、更に、

前記所定のブロックに関連する前記マーカーによって表わされる前記ステータスの少なくとも一部に基づいて、前記第2のオブジェクトが前記追加メモリ部に含まれるか否かを判定する工程を備えることを特徴とするコンピ

ュータ実現方法。

【請求項14】 第1部分と、第2部分と、前記第2部分の要素に関連するワードのアレイと、を含み、前記ワードが、その関連要素の内容が前記第1部分のオブジェクトを参照しているか否かを示すように用意されるコンピュータメモリ内で、所定のオブジェクトを割り当てるためのコンピュータ実現方法であって、

前記第1部分が、前記所定のオブジェクトを割り当てるのに適しているかどうかを判定する工程と、

10 前記第1部分が前記所定のオブジェクトを割り当てるのに適していると判定された場合には、前記所定のオブジェクトを割り当てる工程と、

前記第1部分が前記所定のオブジェクトを割り当てるのに適していないと判定された場合には、前記第1部分でガーベジコレクションを実行する工程であって、前記ガーベジコレクションを実行する工程が、前記第2部分内の関連要素が前記第1部分内の特定のオブジェクトを参照することを表わす所定のワードの位置決めを行うようにワードの前記アレイを走査する工程を備え、前記ガーベジコレクションの実行中、前記第1部分内の前記特定のオブジェクトを保持する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項15】 請求項14に記載のコンピュータ実現方法であって、

前記第2部分内の関連要素が前記特定のオブジェクトを参照することを表わす前記所定のマーカーが位置決めされた場合に、前記第1部分でガーベジコレクションを実行する工程は、更に、

30 前記第2部分の関連要素内部で複数のルートの関係を追跡する工程を備え、前記複数のルートの関係を追跡する工程が、前記コンピュータメモリ内部の新しい位置に前記複数のルートを更新して、前記所定のワードを更新する工程を備えることを特徴とするコンピュータ実現方法。

【請求項16】 請求項15に記載のコンピュータ実現方法であって、

前記所定のワードを更新する工程は、

40 前記複数のルートの中の所定のルートが前記第2部分に存在し、前記第1部分内の前記特定のオブジェクトのうちの選択されたオブジェクトを参照する場合を判定する工程と、

前記所定のルートが前記第2部分に存在し、前記第1部分内の前記選択された特定のオブジェクトを参照する場合には、前記所定のルートが前記第2部分に存在し、前記選択された特定のオブジェクトが前記第1部分に存在することを示す前記所定のワードを更新する工程と、
を備えることを特徴とするコンピュータ実現方法。

【請求項17】 請求項14に記載のコンピュータ実現方法であって、

50 前記第2部分でガーベジコレクションを実行する工程を

更に備え、前記ガーベジコレクションを実行する工程は、前記要素のうちの所定の要素に関連するルートの関係を追跡する工程を備え、前記ルートが新世代に関連し、前記ルートの関係を追跡する工程は、ワードの前記アレイに含まれるワードのいくつかを更新する工程を含むことを特徴とするコンピュータ実現方法。

【請求項18】 第1メモリ部と、複数の要素として用意される第2メモリ部と、を備えるコンピュータシステムにおいて用いられ、前記複数の要素から選択された第1要素と関連するマーカーであって、前記選択された第1要素が前記第1メモリ部内部に含まれるオブジェクトを参照するか否かを認定するように用意される第1インディケータを備えることを特徴とするマーカー。

【請求項19】 請求項18に記載のコンピュータシステムにおいて用いられるマーカーであって、更に、前記オブジェクトへのリファレンスを含み、前記オブジェクトへのリファレンスが、前記オブジェクトに関するアドレスを含むアレイに対するポインタであることを特徴とするマーカー。

【請求項20】 コンピュータシステムであって、第1メモリ部と、複数のブロックに分割される第2メモリ部とを含むメモリと、マーカーのアレイであって、各々のマーカーが前記第2メモリ部内の関連ブロックに対応し、前記マーカーが、それら関連ブロックが関連ブロック外部のオブジェクトを参照する場合を表わすように配置されているマーカーの前記アレイと、ルート・アレイの集合であって、前記ルート・アレイが、前記オブジェクトのアドレスを保持するように用意され、マーカーの前記アレイから選択された所定のマーカーが前記ルート・アレイの集合に関連するような前記ルート・アレイの集合と、前記第1メモリ部内のスペースを回収するための第1ガーベジコレクタと、マーカーの前記アレイに含まれるマーカーのステータスの少なくとも一部に基づいて、前記第2メモリ部内のスペースを回収するための第2ガーベジコレクタと、を備えるコンピュータシステム。

【請求項21】 請求項20に記載のコンピュータシステムであって、マーカーの前記アレイから選択される第1マーカーが、前記第1マーカーの前記関連ブロックが前記第1メモリ部内の第1オブジェクトを参照する場合を認定する部分インディケータを含むことを特徴とするコンピュータシステム。

【請求項22】 請求項20および請求項21のいずれか一方に記載のコンピュータシステムであって、前記複数のブロックが複数のカーとして用意され、前記複数のカーが更に複数のトレインとして用意されること

を特徴とするコンピュータシステム。

【請求項23】 コンピュータシステムであって、複数のブロックに分割されるメモリと、マーカーのアレイであって、各々のマーカーが前記メモリ内の関連ブロックに対応し、前記マーカーが、それら関連ブロックが関連ブロック外部のオブジェクトを参照する場合を表わすように配置されているマーカーの前記アレイと、

ルート・アレイの集合であって、前記ルート・アレイが、前記オブジェクトのアドレスを保持するように用意され、マーカーの前記アレイから選択された所定のマーカーが前記ルート・アレイの集合に関連するような前記ルート・アレイの集合と、

マーカーの前記アレイに含まれるマーカーのステータスの少なくとも一部に基づいて、前記メモリ内のスペースを回収するためのガーベジコレクタと、を備えるコンピュータシステム。

【請求項24】 コンピュータシステムに関連するメモリを動的に管理するためのコンピュータプログラム製品であって、

前記コンピュータの第1メモリ部内で第1のガーベジコレクションを実行するためのコンピュータコードと、前記コンピュータの第2メモリ部上で第2のガーベジコレクションを実行するためのコンピュータコードであって、前記第2メモリ部が、複数のブロックに分割され、前記第2のガーベジコレクションは、前記複数のブロックから選択された所定のブロックに関連するマーカーによって表わされるステータスの少なくとも一部に基づいて、前記所定のブロックが、該所定のブロックに含まれない第2のオブジェクトを参照するところの第1のオブジェクトを含むか否かを判定するコンピュータコードと、

前記コンピュータコードを格納するコンピュータ読み取り可能な媒体と、を備えることを特徴とするコンピュータプログラム製品。

【請求項25】 コンピュータシステムの処理装置に前記コンピュータシステムに関連するメモリを動的に管理させるためのコンピュータプログラム製品であって、前記コンピュータの第1メモリ部内で第1のガーベジコレクションを実行するためのコンピュータコードと、前記コンピュータの第2メモリ部上で第2のガーベジコレクションを実行するためのコンピュータコードであって、前記第2メモリ部が、複数のブロックに分割され、前記第2のガーベジコレクションは、前記複数のブロックから選択された所定のブロックに関連するマーカーによって表わされるステータスの少なくとも一部に基づいて、前記所定のブロックが、該所定のブロックに含まれない第2のオブジェクトを参照するところの第1のオブジェクトを含むか否かを判定するコンピュータコード

と、
搬送波内に具現化される、前記コンピュータコードを表
わすコンピュータデータ信号と、
を備えることを特徴とするコンピュータプログラム製
品。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、一般的にいえば、
コンピュータシステムにおいて動的に割り付けられるメ
モリの管理に関する。更に詳しくいえば、本発明は、自
動記憶領域回収を局部的に実行可能なように、コンピ
ュータシステムに関連するメモリの異なった部分間にお
けるリファレンス（参照）を追跡する技術に関する。

【0002】

【従来の技術】コンピュータシステムに関連するメモ
リの容量は、普通限られている。このため、メモリの保
存と再利用が通常必要となる。ソフトウェア開発者は、種
々のコンピュータプログラミング言語を利用して、コン
ピュータシステム内での動的なメモリ割付を実行するこ
とができる。いくつかのプログラミング言語では、先に
割り付けられたメモリの領域解放を明示的に行う必要が
あるが、この操作は複雑でエラーにつながりやすい。明
示的な手動でのメモリ管理を必要とする言語には、例え
ば、Cプログラミング言語やC++プログラミング言語
がある。また、別のプログラミング言語では、自動記憶
領域回収技術を利用して、メモリを割り付けるコンピ
ュータプログラムを適正に作動させるためにもはや必要で
はないメモリを回収システムから回収する。このような
自動記憶領域回収システムでは、先にメモリを使ってい
たコンピュータプログラムからの明示的な命令やコール
なしにメモリの回収が行われる。

【0003】オブジェクト指向性、即ち、オブジェク
トに基づくシステムでは、メモリ割り当ての典型的な単位
は、当業者に周知のように、通常、オブジェクトあるい
はメモリオブジェクトと称されている。使用中のオブジ
ェクトは、通常、「有効」オブジェクトと称され、一
方、コンピュータプログラムを正しく実行するためにも
はや必要とされないオブジェクトは、通常、「ガーベジ
（ごみ）」オブジェクトと称される。また、ガーベジオ
ブジェクトを回収する動作は、通常、ガーベジコレクシ
ョン（ごみ集め）と称される。更に、自動記憶領域回収
システムを、ガーベジコレクタ（ごみ収集装置）と称す
る。自動記憶領域回収システムを利用するコンピュータ
プログラムは、実行中に有効なメモリオブジェクトを変
更可能であるということから、ミューテーター（可変誘
導装置）と称される。（カリフォルニア州パロアルトの
サンマイクロシステムズ社により開発された）Java
（商標）プログラミング言語やSmalltalkプロ
グラミング言語等の言語で書かれたコンピュータプログ
ラムは、ガーベジコレクションを実行することにより、

自動的にメモリの管理を行っている。

【0004】オブジェクトには、通常、別のオブジェク
トへのリファレンスが含まれている。従って、ガーベジ
コレクタによって管理されるコンピュータメモリの領域
には、通常、互いに参照しあうオブジェクトの集合が含
まれる。図1は、オブジェクトを含むコンピュータメモ
リの領域を例示する図である。通常コンピュータシステ
ムに関連するヒープ（かたまり）であるメモリ10の管
理領域にオブジェクト20が含まれている。一つのオブ
ジェクト20は、一般に、別のオブジェクト20によっ
て参照されている。例えば、オブジェクト20aは、オ
ブジェクト20bに対するポインタ24aを有する。また、
オブジェクト20cは、オブジェクト20bに対す
るポインタ24bを有する。従って、オブジェクト20
bは、オブジェクト20a並びに20cによって参照さ
れる。

【0005】メモリ10には、多くの外部リファレンス
が入力される。図示するように、メモリ10の外部にあ
る固定ルート30には、メモリ10内に位置するオブジ
ェクト20aや20c等のオブジェクトに対するポイン
タ34が含まれる。例えばオブジェクト20aないし2
0dのように、固定ルート30からのリファレンスをた
どることによって到達可能なすべてのオブジェクトが有
効オブジェクトとみなされる。一方、固定ルート30か
らのリファレンスをたどっても到達できないオブジェク
ト20eは、ガーベジオブジェクトと特徴づけられる。

【0006】通常、ガーベジコレクタは、オブジェクト
20e等のガーベジオブジェクトを認定するように実現
される。一般に、ガーベジコレクタは、多くの異なった
アルゴリズムを利用して、処理を実行する。従来用いら
れているガーベジコレクション用のアルゴリズムとして
は参照数計測コレクタ、マーク掃出しコレクタ、複写
コレクタ等が挙げられる。当業者に周知のように、ガー
ベジコレクションを実行中に、オブジェクト20が移動
した場合には、それに従って、オブジェクト20へのリ
ファレンスを調整しなければならない。

【0007】管理されるメモリ領域を小さな部分に分け
て、一時に一つの領域で局在的にガーベジコレクション
を実行可能なようにする構成が望ましい場合がある。メ
モリ区分案の一つが、世代ガーベジコレクションであ
る。世代ガーベジコレクションでは、オブジェクトが生
成された時点から測定される寿命に基づいて、オブジェ
クトを分ける。「若い」オブジェクトは、「年寄り」オ
ブジェクトに比べて、ガーベジになる可能性が高いこと
が知られている。したがって、世代ガーベジコレクシ
ョンを利用することによって、メモリ回収の全体的な効率
を上げることができる。

【0008】図2は、新世代と旧世代とに区分されるメ
モリとルートとの間のインターフェースを例示する図で
ある。通常コンピュータシステムに関連するヒープであ

るメモリ110には、新世代110a並びに旧世代110bが含まれる。固定ルート114、即ち、新世代110aと旧世代110bのいずれかあるいは両方に含まれるオブジェクトを参照する全体的オブジェクトは、図示するように、新世代110aに含まれるオブジェクト120に対するポインタ116を備える。当業者に周知のように、ルート114は、スタック上に位置するものでもよい。

【0009】新世代110a内のいくつかのオブジェクト120、例えば、オブジェクト120aをルートとみなすこともできる。これは、オブジェクト120aが、有効オブジェクトであると考えられ、他のオブジェクト120dに対するポインタ122を備えるためである。新世代オブジェクト126が有効オブジェクトであり、旧世代オブジェクト128を参照している場合、旧世代110b内で実行されるガベージコレクションでは、通常、オブジェクト128が「回収」されない。一方、新世代オブジェクト126が無効オブジェクトの場合、旧世代オブジェクト128は他のオブジェクトによって参照されていないため、新世代110a内で実行されるガベージコレクションでは、旧世代オブジェクト128が到達不能となる。旧世代オブジェクト128が到達不能であれば、旧世代110bで実行されるガベージコレクションの結果、旧世代オブジェクト128の回収が行われる。ここで、新世代オブジェクト126と旧世代オブジェクト128との間をつなぐポインタ130は、新世代110aと旧世代110bの両方に伸びているため、世代間ポインタとみなされる。ポインタ130が新世代オブジェクト126から旧世代オブジェクト128に伸びている場合、新世代のガベージコレクションでは旧世代オブジェクト128を回収不可能であるため、旧世代オブジェクト128は保有ガベージとみなされる。

【0010】また、別のメモリ区分案では、一回のガベージコレクションを実行するために必要な時間を短縮するように、メモリを小さな領域に分けている。ガベージコレクションに起因する中断は、しばしば、関連するミューテーターの中断につながり、望ましくない。あるシステムでは、ガベージコレクタに充分小さい最大許容中断期間が設定されている。このようなガベージコレクタは、実時間ガベージコレクタとして知られている。また、別のシステムでは、中断時間を短くするようにガベージコレクタが働くが、所定の状況下ではこの試みが失敗に終わることもある。中断時間を短くするように働くガベージコレクタは、非中断あるいは増分ガベージコレクタとして知られている。

【0011】個々のメモリ領域上で処理を実行するためには、ガベージコレクタが、その領域内に伸びるすべてのリファレンスに関する知識を持っている必要がある。領域内に伸びるリファレンスは、その領域に対するルートと称される。ここで、ルートには、固定ルートのように

な外部リファレンスと、コンピュータメモリの他の領域からのリファレンスと、の両方が含まれる。したがって、ガベージコレクタは、通常、ルート、即ち、リファレンスを見つけて、追跡するためのメカニズムを備える。

【0012】メモリ領域内でリファレンスの位置決めを行うために可能な方法の一つは、メモリ内のすべてのオブジェクトを走査する方法である。しかし、大部分のシステムでは、メモリ内のすべてのオブジェクトを走査する方法は、時間がかかるため好ましくない。したがって、もっと効率的な追跡案が求められている。

【0013】ミューテーターがオブジェクトにリファレンスを記憶させると、ガベージコレクション（ごみ集め）を目的としてリファレンスを追跡するように追加処理が実行される。この追加処理では、バリア（障壁）の書き込み、即ち、チェックの記憶が行われる。ミューテーターの効率を許容可能なレベルに維持するために、バリアの書き込みに関連するコストをできるだけ低く押さえる必要がある。

【0014】メモリ領域内でリファレンスを追跡するための方法の一つは、その領域に対するすべてのルートを保有する集合を保持する方法である。このような集合は、一般に、その領域に対する記憶集合として知られている。関連するバリアの書き込みにより、ある領域へのリファレンスが記憶された時を検出することができる。したがって、バリアの書き込みにより、所定の領域に関連する記憶集合内でのリファレンスの位置が挿入される。

【0015】図3は、新世代に含まれるオブジェクトと旧世代に含まれるオブジェクトとをつなぐポインタで、記憶集合を利用して追跡されるポインタを例示する図である。メモリ302は、新世代302aと旧世代302bとに分割される。記憶集合304を用いて、旧世代オブジェクト310から新世代オブジェクト312に伸びるポインタ314を追跡する。旧世代オブジェクト310aには新世代オブジェクト312bに対するポインタ314aが含まれるので、旧世代オブジェクト310aのアドレスは、記憶集合304に格納される。同様に、新世代オブジェクト312bと312aに対するポインタ314b及び314cを含む旧世代オブジェクト310bのアドレスも、記憶集合304に格納される。

【0016】記憶集合にはルートの全てが含まれているため、記憶集合を用いればガベージコレクションを目的としたルートの位置決めが確実に実行される。ただし、バリアの書き込みを利用する方法は、余分のメモリを必要とするため、高くつく場合がある。また、新しい位置が記憶集合に挿入された場合、その位置が記憶集合内にすでに存在している可能性もある。ある位置を挿入する前に記憶集合にその位置がすでに存在していないかどうかをチェックする方法は、高くつく。その一方で、ある

位置が重複していないかどうかのチェックを省けば、記憶集合が不必要に大きくなってしまいう可能性がある。当業者に周知のように、一般に、記憶集合が保有できる重複の数には上限がない。所定の位置に対するリファレンスを格納する時に、その位置が、すでに、先のリファレンスで参照されている場合がある。この場合、格納操作に先立って実行された所定の位置に関係する記憶集合への入力、後で行われる格納操作によって不必要になる。前の入力を解除する処理は、状況によって、費用のかかる操作となるが、その一方、前の入力をそのまま放置しておけば、記憶集合が余分のメモリを占有することにつながる。

【0017】メモリ領域内でリファレンスを追跡するために用いられる別の案にカードマーキング法がある。一般に、カードマーキング法では、カードと呼ばれる比較的小さな部分にメモリを概念的に分割する。ガーベジコレクタは、カード一枚ごとに一つの入力となるように、ビット・アレイ（列）の割り当てを行う。リファレンスが格納されると、バリアの書き込みを実行することにより、対応するカード・アレイの入力を算出して、関連ビットを設定する。この処理は、カードの「汚染」と呼ばれている。ここで、効率を考えて、ビット・アレイの代りにバイト・アレイやワード・アレイを用いることもできる。

【0018】カードマーキングを用いる利点の一つは、バリアの書き込みが比較的安価であることである。また、カードマーキングの別の利点は、カードアレイに必要なメモリ容量が一定であることである。ただし、ガーベジコレクションの実行時にルート位置を決めるために必要な処理が複雑で、記憶集合を用いたシステムに比べてより多くの処理を必要とする場合がある。この方法では、ガーベジコレクタは、リファレンスの格納が行われた大体の場所を知っているだけである。即ち、ガーベジコレクタは、カードマーキング・アレイの汚染入力があった場所を知っているだけである。したがって、汚染入力が認定されると、対応するカードを走査してルートを探索する必要がある。ルートを見つけるためにカードを走査する方法は、メモリ全体を走査するよりもずっと安価であるが、カードの走査は、それでも、費用のかかる方法である。当業者に周知のように、ガーベジコレクタがルートを位置決めすると、次のガーベジコレクションでルートの位置決めを実行するために、対応するカードアレイの入力を汚染状態のまま保持する必要がある。

【0019】また、記憶集合とカードマーキングの両方を用いる合成案とすることもできる。バリアの書き込みにより、上述したようなカードマーキングを実行する。ただし、この方法では、ガーベジコレクションの実行時に、ガーベジコレクタがカードごとに記憶集合を作成する。ガーベジコレクションが完了すると、カードマーキング・アレイがクリアされる。記憶集合とカードマーキ

ングの合成案を用いることにより、後に続くガーベジコレクションのために必要な走査量を削減することができ、その結果、ガーベジコレクション処理の全体的な効率を上げることができる。ただし、このような案の実行は、複雑である場合が多い。このため、記憶集合とカードマーキングの合成案を効率よく実現する方法並びに装置が求められていた。

【0020】

【課題を解決するための手段およびその作用・効果】本発明は、コンピュータメモリ内で世代ガーベジコレクションを実行するための方法並びに装置に関する。本発明の方法は、第1メモリ部と、関連マーカを各々有する複数のブロックに分割される第2メモリ部と、を含むメモリを、動的に管理するためのコンピュータ実現方法であり、第1メモリ部上で、第1のガーベジコレクションを実行する工程を備えることを要旨とする。また、本発明の方法は、第2メモリ部の複数のブロックのうちの所定のブロック上で、第2のガーベジコレクションを実行する工程を備える。さらに、第2メモリ部の所定のブロック上で、第3のガーベジコレクションを実行する。第3のガーベジコレクションは、所定のブロックに関連するマーカによって表わされるステータスの区少なくとも一部に基づいて、所定のブロックが、その所定のブロック内に含まれていない第2のオブジェクトを参照するところの第1のオブジェクトを含むか否かの判定を含む。ステータスは、第2のガーベジコレクションの実行後に第2のオブジェクトへのリファレンスが記憶されたか否かの表示を含む。第2のガーベジコレクションの実行後に第2のオブジェクトへのリファレンスが記憶された場合には、所定のマーカを用いて、新たなルート・アレイが生成される。

【0021】本発明の別の方法は、コンピュータメモリ内で所定のオブジェクトを割り当てるためのコンピュータ実現方法であって、メモリの第1部分が、所定のオブジェクトを割り当てるのに適しているかどうかを判定する工程と、第1部分が所定のオブジェクトを割り当てるのに適していると判定された場合に、所定のオブジェクトを割り当てる工程と、を備えることを要旨とする。また、上記方法は、第1部分が所定のオブジェクトを割り当てるのに適していないと判定された場合には、第1部分でガーベジコレクションを実行する工程を含む。第1部分でガーベジコレクションを実行する工程は、所定のワードに関連する第1ルートがメモリ内にすでに存在するオブジェクトを参照することを示すように配置された所定のワードの位置決めを行うように、ワードのアレイを走査する工程を含む。

【0022】更に、本発明のコンピュータシステムは、第1メモリ部と、複数のブロックに分割される第2メモリ部と、して用意されたメモリを含むことを要旨とする。本発明のコンピュータシステムは、更に、メモリに

接続されるプロセッサと、第1メモリ部に関連する第1ガーベジコレクタを含む。第2メモリ部に関連する第2ガーベジコレクタは、所定のブロックがその所定のブロックに含まれていない第2のオブジェクトを参照する第1のオブジェクトを含むか否かを示すように配置された関連マーカーを有する所定のブロック上でガーベジコレクションを実行するように構成される。所定のマーカーは、所定のブロックに含まれていない第2のオブジェクトを所定のオブジェクトが参照する場合に、そのオブジェクトのアドレスを保持するルート・アレイに関連する。1つの実施例では、第2ガーベジコレクタは、マーカーを用いて、第1メモリ部内に第2のオブジェクトが含まれるか否かを認定するように、構成される。

【0023】本発明の上記並びに他の利点は、以下の詳細な説明を読み、図面を研究することによって明らかになるであろう。

【0024】

【発明の実施の形態】本発明並びにその更なる利点は、添付する図面とともに以下の説明を参照することにより、理解されるであろう。

【0025】ガーベジコレクションは、コンピュータプログラムによって用いられるコンピュータメモリの動的な割り当てを行うものである。一般に、ガーベジコレクションは、プログラムの実行を中断させる。ガーベジコレクションに関連する中断時間を短くするために提案された装置では、メモリが複数の世代に分割されている。例えば、二世代メモリ構造において、メモリを、新たに割り当てられたオブジェクトを含む新世代と、古いオブジェクトを含む旧世代と、に分割するようにしてもよい。ここで、新たに割り当てられたオブジェクトは、古いオブジェクトよりも早く無効になる可能性が高い。したがって、ガーベジコレクションは、旧世代よりも新世代で頻繁に実行される。通常、メモリ全体の大きさに比べて、新世代はかなり小さい。したがって、プログラムの実行中に実質的な中断を引き起こすことなく、新世代におけるガーベジコレクションを行うことができる。

【0026】新世代におけるガーベジコレクションが有意な中断を引き起こさない一方、旧世代におけるガーベジコレクションはプログラムの実行中に有意な中断を引き起こす。これは、旧世代が、通常、新世代よりも大きいことに起因する。更に、旧世代に含まれるオブジェクトから新世代に含まれるオブジェクトに伸びる世代間ポインタが存在する場合には、プログラムの実行中に、世代間ポインタの追跡が行われる。このようなポインタに関連する旧世代のオブジェクトは、新世代のルートであると考えられるため、世代間ポインタの追跡が行われる。旧世代に含まれるオブジェクトから新世代に含まれるオブジェクトへのポインタ、即ち、リファレンスの追跡は、複雑で、時間がかかることが多い。

【0027】旧世代のメモリを小さな領域、即ち、プロ

ックに分割することによって、旧世代のガーベジコレクションに関連する中断時間を減らすことができる。また、旧世代ガーベジコレクションを、毎回、旧世代メモリの所定のブロック内で実行するようにしてもよい。旧世代ガーベジコレクションが実行されるブロックを制限し、かつ、旧世代ガーベジコレクションが実行されるたびに異なったブロックでガーベジを回収するようにすれば、旧世代ガーベジコレクションの効率を増大させることができる。

10 【0028】前述したように、旧世代メモリは、旧世代ガーベジコレクションを容易にするように、ブロックに分割される。旧世代ガーベジコレクションは、毎回、任意の適当なブロック上で実行される。即ち、旧世代ガーベジコレクションを漸増方向で実行するようにしてもよい。例えば、ガーベジコレクションがまだ行われていないブロックのうち最も古いブロックでガーベジコレクションを実行するようにしてもよい。あるいは、ガーベジコレクションが既に行われたブロックのうち一番先に行われたブロックでガーベジコレクションを実行するようにしてもよい。図4を参照して、旧世代におけるメモリのブロックへの分割を、本発明の一実施例にしたがって説明する。旧世代404は、メモリのブロック408を含む。メモリのブロックは、通常、必要に応じて割り当てられる。即ち、実質的にすべての既存ブロック408が一杯になると、追加ブロックの割り当てが行われる。所定のプログラムに関連するオブジェクトを含むブロック408は、「カー」とも称される。ブロック、即ち、カー408は、「トレイン」412と称される連結された集合として配置される。例えば、トレイン412aは、カー408a及び408bを、トレイン412bは、カー408c、408d、408eを、また、トレイン412cは、カー408f及び408gを含む。

20 【0029】トレイン412を構成するカー408は、一般に、所定のカー408が所定のトレイン412に加えられた時に基づく順に並べられる。例えば、トレイン412aにおいて、カー408bがカー408aの後でトレイン412aに加えられた場合、カー408aは、カー408bと同じトレインの低位のカーと考えられる。同様に、トレイン412は、所定のトレイン412が形成された時に基づく順に並べられる。図示するように、トレイン412bがトレイン412cの後でメモリ404に加えられた場合、トレイン412cは、トレイン412bに対して下位のトレインであり、その結果、トレイン412b内のカー408c、408d、408eに対して下位のトレインになる。

30 【0030】トレイン412を用いて、旧世代404内でガーベジコレクションを実行するアルゴリズムは、通常、一時に、所定の数のカー408を回収する。実施例では、各回の旧世代ガーベジコレクションの間に、たった一つのカー408のみが回収される。ただし、各回の

旧世代ガーベジコレクションの間に回収されるカーの数はこれに限定されるものではなく、任意の適当な数を回収することができる。一時に一つのカー408のみを回収することによって、旧世代ガーベジコレクションの全体的な長さ、すなわち、旧世代ガーベジコレクションに関連する中断時間が基本的に制限される。このように、各回の旧世代ガーベジコレクションの実行中に所定数のカー408のみを回収することによって、旧世代ガーベジコレクションに通常関連する長い中断時間を削減することができる。

【0031】一般に、カー408でガーベジコレクションを実行する場合、旧世代メモリ内の各カー408を起点とする、あるいは、各カー408内のオブジェクトにアクセスするすべてのポインタの追跡が行われる。ポインタを追跡する方法を図5を参照して以下に説明する。回収すべき所定のカー408が有効オブジェクトを含む場合、そのオブジェクトを指し示す、あるいは、そのオブジェクトによって指し示されるトレイン412内にそのオブジェクトを複製する。オブジェクトの複製が完了すると、「もとの」オブジェクトが回収される。所定のトレイン412が外部に通じるポインタをまったく持たない場合、そのトレイン412は消去可能である。すなわち、所定のトレイン412に割り当てられたメモリが解放される。同様に、所定のカー408が外部に通じるポインタをまったく持たない場合、そのカー408は消去可能である。

【0032】実施例において、所定のトレイン412内のカー408の大きさは一定である。即ち、すべてのカー408が実質的に同じ大きさである。一つのカー408がリンクしたオブジェクトを収容するのに十分な領域を有していない場合、一つのカー408、例えばカー408f内のオブジェクト416aを第2のカー408、例えばカー408g内のオブジェクト416bにリンクさせるようにしてもよい。オブジェクト416a及び416bは異なったカー408内に存在し、互いに参照しあっているため、旧世代ガーベジコレクションの実行中、オブジェクト416a及び416bは有効であり続ける可能性が高い。このように、旧世代ガーベジコレクションを実行してもカー408f及び408gが回収されない結果となる場合もある。トレイン412c内部でカー408fと408gとを結ぶポインタが追跡されない場合、カー408f及び408gに関連するメモリは、通常解放されない。

【0033】一般的にいうと、トレイン412の長さ、並びに、カー408の大きさは様々に変更可能である。カー408の大きさは、これらに限定されるものではないが、旧世代404の大きさ、ガーベジコレクションアルゴリズムの速度、およびガーベジコレクションアルゴリズムを実行するコンピュータの速度等の要因に依存する。例えば、現在のシステムで良好に作動するカーの大

きさは、約20キロバイトのメモリサイズから約100キロバイトのメモリサイズまでである。実施例では、カー408のメモリサイズは約64キロバイトである。

【0034】カー408は更にカード420に分割することが可能である。カード420を用いることによって、カー408上で実行される旧世代ガーベジコレクションを容易にすることができる。これに関しては、図5を参照して後述する。カー408と同様に、カード420も任意のサイズを持つことができる。ただし、カード420のサイズは、通常、約100ないし1000バイトである。例えば、カー408が約64キロバイトのメモリサイズを有する場合、カード420が約512バイトのサイズを持つようにしてもよい。

【0035】旧世代内の異なった領域に関する記憶、即ち、マーカーを準備することによって、旧世代オブジェクトから新世代オブジェクトにつながる世代間ポインタの追跡効率を向上させることができる。例えば、マーカーは、所定の領域の関連部分が新世代内のオブジェクトに対するポインタを含んでいるか否かを示すように設定されるフラグを含むワードである。このようなワードを用いることによって、旧世代内のオブジェクトから新世代内のオブジェクトへの世代間ポインタの追跡を効率よく行うことができる。ここでマーカーの例としてワードを挙げたが、マーカーは、ビット、バイト等、任意の適当な記憶形態であればよい。

【0036】旧世代メモリ内のカードに関連するオブジェクトから新世代メモリ内のオブジェクトにつながる世代間ポインタを追跡するために、カードが「追跡構造」を備えるようにしてもよい。追跡構造、即ち、カードマーカーを用いることによって、あるカードに関して世代間ポインタが存在しているか否かを認定し、また、そのカード内部のいずれのオブジェクトがポインタを含んでいるかを認定することができる。図5は、本発明の一実施例にしたがって、カードマーカー・アレイと関連するルート・アレイのプールとを例示する図である。ルート・アレイは、通常、あるカードに対するオフセットとして指定されるルート位置を含む小さなアレイである。言い換えると、ルート・アレイは、基本的に、小さな記憶集合である。カードマーカー452のアレイ450は、ルート・アレイ460のプール464に含まれるルート・アレイ460に対するポインタ456を含む。カードマーカー452は、通常、トレイン内部のカーに関連する。更に詳しく言えば、トレイン内部のカーを、旧世代オブジェクトを保持する多数のカードに分割することができる。このように、トレイン内部のカーは、多くのカードマーカー452に関連づけられる。あるカーに係るすべてのカードマーカー452を含む場合でも含まない場合でも、カードマーカー452のグループがカードマーカー452のアレイ450であると考えられる。実施例では、旧世代に関連するすべてのカードマーカー452がアレイ4

50に含まれる。

【0037】実施例では、例えばカードマーク452a等のカードマーク452は、32ビットのワードである。ただし、一般に、カードマーク452aは、任意の適当な長さのワードであればよい。あるいは、カードマークを構成するデータ構造を多数の成分に分割してもよい。カードマーク452aを構成するビットを所定の適用に適した任意の方法で割り当てることができるが、例えば、カードマーク452aの最下位ビットLSBをカードマーク452aの「状態」、即ち、全体的な条件を基本的に認定するフラグとして用いる。カードマーク452aの最下位ビット468は「汚染」フラグである。即ち、最下位ビット468は、一般に、カードマーク452aが、より詳しく言えば、カードマーク452aに関連する(図4に示すような)カードが「汚染されている」か否かを示すように構成される。ここで、汚染カードは、前回ガーベジコレクションを実行して以来ポイントが格納されているカードである。このようなポイントには、限定されるものではないが、旧世代メモリから新世代メモリにつながる世代間ポイント、下位トレインに対するポイント、カードマーク452aに関連する同一トレイン内の下位のカーに対するポイントが含まれる。最下位ビット468が設定されると、即ち、最下位ビット468が「0」に設定されると、カードマーク452aが汚染されていることが示される。

【0038】カードマーク452aの最下位から二番目のビット470は、新世代フラグであり、カードマーク452aが新世代に対する世代間ポイントを含むか否かを示すために用いられる。最下位から二番目のビット470が設定されると、即ち、「0」に設定されると、カードマーク452aが新世代に対する世代間ポイントを含むことが示される。最下位から三番目のビット472は、実施例において、下位トレインフラグである。下位トレインフラグは、カードマーク452aが下位トレインに対するポイントの有するか否かを示すように構成される。下位トレインフラグ、即ち、最下位から三番目のビット472が設定されると、カードマーク452aが下位トレインに対するポイントを含むことが示される。最下位から四番目のビット474は、カードマーク452aが同一トレイン内のより下位のカーに対するポイントを含むか否かを示すように構成される同一トレインフラグである。

【0039】実施例では、カードマーク452aがクリーンであると考えられる場合、カードマーク452aは、通常、旧世代から新世代につながるポイント、下位トレインに対するポイント、あるいは、同一トレイン内のより下位のカーに対するポイントを含まない。このように、カードマーク452aがクリーンな場合、最下位ビット468、最下位から二番目のビット470、最下位から三番目のビット472、ならびに、最下位から四

番目のビット474は、すべて値「1」に設定されている。言い換えると、カードマーク452aがクリーンである場合には、カードマーク452aの最下位4ビットがクリアされている。したがって、図示するように、カードマーク452a及び452bはクリーンであり、最下位ビットが設定されたカードマーク452cは汚染されている。

【0040】カードマーク452aに含まれる残り28個のビットを、カードマーク452aに関連するポイントのアドレスを基本的に保持する関連ルート・アレイ460を示すポイントとして利用されるように割り当ててもよい。あるいは、例えば、実施例のように、カードマーク452aが27ビットのポイント478と予備ビット480を含むようにしてもよい。27ビットのポイント478は、カードマーク452aからルート・アレイ460aにつながるポイント456aを認定する。ルート・アレイ460aは、8ワードあるいは16ワードで構成されていることが多い。ただし、ルート・アレイ460aを構成するワードの数は様々に変更可能である。ルート・アレイ460は、通常、ルートに対するリファレンスを保持するように構成される。例えば、ルート・アレイ460aは、カードマーク450aに関連するルート、したがって、カードマーク450aに関連する(図4に示すような)カードを保持する。

【0041】適当なルート・アレイ460がいくつかの場合、「ルート・アレイのオーバーフロー」が起こると考えられる。所定のルート・アレイ460に格納すべきルートが過剰に多い場合、もはやルートの追跡を行うことができない。プール464内のルート・アレイ460に対するポイントの容量が、プール464内で割り当てられた利用可能なルート・アレイ460に対して過剰な場合、ルート・アレイのオーバーフローが起こる。この結果、ガーベジコレクションを実行すると、ルート・アレイのオーバーフローに関係するカード内のポイントが検索されて、ルートの認定が行われる。ルート・アレイのオーバーフローが起こると、27ビットのポイント478が零に設定される。即ち、27ビットのポイント478に含まれる27個のビットが「0」に設定される。

【0042】ガーベジコレクション処理に直接関係する、あるいは、直接関係しない任意の数の適当な目的のために予備ビット480を用いることができる。ここで、適当な目的とは、以下に限定されるものではないが、例えば、予備ビット480を用いてカードマーク452aの年齢(経過期間)を示すものである。また、カードマーク452aに含まれるポイント、例えば、27ビットのポイント478に関連するビットの数を減らすことによって、追加の予備ビットがカードマーク452aに含まれるようにしてもよい。

【0043】カードマークは、新世代に含まれるオブジェクトに対する世代間ポイントを含むカー内のカードを

追跡するように、ガーベジコレクション処理に関連して用いられる。例えば、コンピュータプログラムの実行中、カードマークが実質的に現在処理中のカードマークとして維持されるようにカードマークを更新するようにしてもよい。その後のガーベジコレクションの実行中、世代間ポインタを有するカードを認定するように、カードマークがチェックされる。

【0044】コンピュータプログラムの実行中にガーベジコレクション処理を容易に行うためにカードマークを用いる場合、通常、カードマークによって参照されるオブジェクトを認定するように、カードマークが検索され、かつ、更新される。次に、図6を参照して、本発明の一実施例にしたがって、ガーベジコレクション処理を利用するプログラムを実行するためのステップを説明する。所定のプログラムを実行する前に、ステップ502で、そのプログラムに関係するカードアレイを初期化する。例えば、所定のカーに關係するカードアレイには、図5を用いて上述したように、プログラムによって利用されるオブジェクトに關係するカードマーク、あるいは、カードマークに対するリファレンスが含まれる。このように、一般に、カード・アレイは、プログラム内で利用されるオブジェクトを追跡するために用いられる。カード・アレイが初期化された後、オブジェクトを割り当てること、あるいは、オブジェクトに関するポインタを格納することが必要になるまで、ステップ503でプログラムを実行する。オブジェクトを割り当てるために、通常、メモリをオブジェクトに割り当てる処理が行われ、また、オブジェクトに関するポインタを格納するために、他のオブジェクトに対するポインタを第1オブジェクト内に格納する処理が実行される。

【0045】プログラムの実行中、ポインタを格納するための命令が定期的に与えられる。ステップ504で、あるポインタを格納すべきと判定された場合、ステップ512で、ポインタを格納すべきオブジェクトに対応するカード・アレイの入力が認定される。カード・アレイの入力が認定されると、ステップ514で、そのカード・アレイの入力は「汚染されている」とマークされる。カード・アレイの入力、即ち、カードマークが汚染されていることをマークするために、図5を用いて上述したように、入力の汚染フラグ、即ち、入力452aの最下位ビット468を設定する処理が行われる。カードマークが汚染されているとマークされた後、ステップ516で任意の適当な方法を用いてポインタが格納される。ポインタが書き込まれると、処理がステップ503に戻って、プログラムの実行が続けられる。プログラムが終了すると、ステップ502でカードマーク・アレイが初期化され、プログラムの再実行が開始される。

【0046】また、オブジェクトを割り当てるための命令も実行中に与えられる。例えば、ステップ504に戻って、オブジェクトを割り当てる必要がある場合、ステ

ップ506で、割り当てられるべきオブジェクトに対して十分な新世代メモリが存在しているか否かの判定が行われる。ステップ506でオブジェクトの割り当てに利用される十分な新世代メモリがあると判定された場合には、ステップ508でオブジェクトの割り当てが行われ、処理がステップ503に戻って、格納すべきポインタが存在するか否かの判定が行われる。

【0047】逆に、ステップ506で割り当てられるべきオブジェクトに対して十分な新世代メモリがないと判定された場合には、処理は、ステップ510に進んで、ガーベジコレクションが実行される。ガーベジコレクションの実行中、新世代メモリを、また、おそらくは旧世代メモリも、クリーンにして、オブジェクトが割り当て可能なように新世代メモリの一部を解放する。任意の適当な方法でガーベジコレクションを実行することができるが、以下に、図7を参照して、特に好適なガーベジコレクションの方法を説明する。ステップ510でガーベジコレクションを実行した後、処理は506に戻って、割り当てられるべき所定のオブジェクトに対して、即ち、ステップ503で割り当てが必要であると考えられたオブジェクトに対して、十分な新世代メモリがガーベジコレクションで準備されているか否かを判定する。

【0048】図7を参照して、本発明の一実施例にしたがって、ガーベジコレクションの実行に關係するステップを説明する。すなわち、図6のステップ510を詳述する。ガーベジコレクションは、オブジェクトの割り当てが可能になるように十分なメモリ領域を「解放する」ために、新世代メモリと旧世代メモリの両方で実行可能である。ガーベジコレクションが実行されるシステムがマルチスレッドシステムである場合には、任意の適当な方法を用いて、ステップ602でスレッドを同期化させる。同期化が必要なスレッドが同期化されると、ステップ604で、新世代メモリ内でガーベジコレクションが実行される。図8を参照して、新世代ガーベジコレクションを実行する方法に關するステップを後述する。

【0049】ステップ604で新世代ガーベジコレクションを実行した後、処理はステップ606に進み、旧世代メモリに關するガーベジコレクションを実行する必要があるか否かの判定が行われる。実施例では、旧世代ガーベジコレクションが定期的に実行される。即ち、所定の時間が経過した後、旧世代ガーベジコレクションを実行する。ただし、旧世代ガーベジコレクションが必要か否かの判定に、他の、あるいは、追加の基準を用いることもできる。

【0050】旧世代ガーベジコレクションが必要であると判定された場合、ステップ608で、カードマーキング案を用いて旧世代ガーベジコレクションを実行する。カードマーキング案を用いて旧世代メモリ上で実行されるガーベジコレクションには様々な方法を適用可能であるが、図14及び図15を参照して、適切な旧世代ガー

10

20

30

40

50

ベジコレクション方法の一つを後述する。

【0051】ステップ608で旧世代ガーベジコレクションが完了すると、オブジェクトの割り当てを可能にするために実行されるガーベジコレクションの全体的な処理が完了する。即ち、処理は、図6のステップ506に進み、割り当てられるべきオブジェクトに対して十分な新世代メモリが利用可能であるか否かの判定が行われる。同様に、ステップ606で旧世代ガーベジコレクションが必要でないと判定された場合にも、処理は、図6のステップ506に進む。

【0052】図8は、本発明の一実施例にしたがって、新世代ガーベジコレクション処理に関するステップ、即ち、図7のステップ604の詳細を示すフローチャートである。まず、ステップ701で、固定ルートの追跡が行われる。前述したように、固定ルートは、ガーベジコレクションによってクリーンにされている新世代メモリあるいは旧世代メモリ内を直接指し示すルート、例えば、スタック上の有効オブジェクトである。固定ルートの追跡は、通常、ルートのリファレンスの推移的な追跡を伴う。固定ルートあるいはその他のルートの追跡に係る実際のステップに関しては、図9ないし図12を参照して、後述する。

【0053】固定ルートの追跡後、ステップ702でカードアレイを走査して、設定された汚染フラグおよび新世代フラグのいずれかあるいは両方を有するカード・アレイ内の次の入力的位置決めを行う。上述したように、例えば、あるビットを「設定する」とは、値「0」をそのビットに割り当ててことを意味し、また、あるビットを「クリアする」とは、値「1」をそのビットに割り当ててことを意味する。次に、ステップ703で、設定された新世代フラグおよび汚染フラグのいずれかあるいは両方を有する入力、即ち、カードマークが見つかったか否かが判定される。このような入力が見つかった場合には、処理がステップ704に進み、その入力に汚染されているか否かの判定が行われる。

【0054】ステップ704で入力に汚染されていると判定された場合には、ステップ707でその入力に関連するカードがクリーンであるとマークする。ここで、カードがクリーンであるとマークするとは、そのカードに係るカードマークがクリーンであるとマークすることを意味する。ステップ707でカードがクリーンであるとマークした後、処理はステップ708に進み、そのカードに含まれるすべてのオブジェクトを走査して、次に追跡されるルートの位置決めを行う。上述したように、ルートの追跡に係るステップに関しては、図9ないし図12を参照して、後述する。ルートの追跡が実行されると、処理は、ステップ702に戻り、設定された汚染フラグおよび新世代フラグのいずれかあるいは両方を有する次の入力、即ち、カードマークの位置決めを行うために、カード・アレイを走査する。

【0055】ステップ704で入力に汚染されていないと判定された場合には、その入力は、設定された新世代フラグを有している。言い換えると、その入力には、世代間ポインタが含まれている。入力に汚染されていなければ、処理がステップ704からステップ706に進み、その入力に係るルート・アレイがオーバーフローしたか否かが判定される。オーバーフローしたルート・アレイとは、追加ルートの挿入を受け入れるのに十分な領域を含んでいないルート・アレイを意味する。ある入力に係るルート・アレイがオーバーフロー状態であると判定された場合には、処理がステップ707に進み、その入力に係るカードがクリーンであるとマークされる。

【0056】処理はステップ714からステップ718に進み、入力に係るブロックが走査され、そのブロックに係るルートが追跡される。次に、処理はステップ702に戻り、カード・アレイの走査を実行して、カード・アレイ内でゼロに設定された最下位ビットを有する次の入力の検索を行う。

【0057】ステップ703で、適当な入力、即ち、ゼロに設定された最下位ビットを有する入力が見つからない場合には、ステップ710で、すべての複製されたオブジェクトの走査が完了するまで、新世代メモリ内でのルートの追跡が行われる。すべての複製されたオブジェクトの走査が完了すると、新世代ガーベジコレクション処理が完了する。

【0058】一般に、ガーベジコレクションには、上述したように、ルートの追跡処理が含まれる。次に、図9を参照して、本発明の一実施例にしたがって、一つのルートを追跡する処理に係るステップを詳述する。処理が開始されると、まず、ステップ720で、ガーベジコレクションが実行されているメモリ領域内をルートが指し示しているか否かが判定される。ルートがガーベジの回収が行われている領域内を指し示していないと判定された場合には、ルートを追跡する処理を終了する。一方、ステップ720で、ルートがガーベジコレクションが実行されている領域内を実際に指し示していると判定された場合には、次に、ステップ721で、カードマークのヘッダ領域が前進ポインタを保持しているか否かの判定が行われる。即ち、カードマークに係る前進ポインタによって新しいメモリ位置が認定されるか否かが判定される。

【0059】ヘッダ領域が前進ポインタを保持していると判定されると、処理は、ステップ721からステップ722に移り、前進カウンタによって認定された新しい位置にルートを更新する。ルートが更新されると、それにしたがって、ステップ728でカードマークが更新される。カードマークの更新処理に関するステップに関しては、図10を参照して、後述する。カードマークを更新後、ルートの追跡処理を終了する。

【0060】ステップ721でヘッダ領域が前進ポイントを保持していないと判定された場合には、ステップ724で、そのルートによって認定されたオブジェクトを新しい位置に複製して、オブジェクトのヘッダに前進ポイントを挿入する。前進ポイントは、オブジェクトの新しい位置を認定する。前進ポイントをヘッダ内に挿入した後、ステップ726で、そのルートを新しい位置に更新する。次に、ステップ728で、関連するカードマークを更新して、ルートの追跡処理を終了する。

【0061】次に、本発明の一実施例にしたがって、図10ないし図12を参照して、カードマークの更新処理に関するステップ、即ち、図9のステップ728を詳述する。まず、ステップ740で、ポイントが旧世代から新世代につながっているか否かの判定が行われる。ポイントが旧世代から新世代につながっている場合には、次に、ステップ746で、旧世代から新世代へのポイントが存在することを示す新世代フラグが設定されているか否かが判定される。

【0062】新世代フラグが設定されているとステップ746で判定された場合、次のステップ752で、ルート・アレイがオーバーフローしたか否かが判定される。ルート・アレイがオーバーフローしている場合には、カードマークを更新する処理を完了する。一方、ルート・アレイがオーバーフローしていない場合には、ステップ754で、ルート・アレイがいっぱいか否かが判定される。ルート・アレイがいっぱいではないと判定されると、処理がステップ750に進み、ルート、即ち、現在追跡されているルートが、そのルート・アレイに挿入され、その後、カードマークを更新する処理を完了する。一方、ステップ754で、ルート・アレイがいっぱいであると判定されると、処理は、ステップ755に進み、ルート・アレイのオーバーフローを示すように、カードマークが設定される。カードマークを設定した後、カードマークの更新処理を完了する。

【0063】一方、ステップ746で新世代ポイントが設定されていないと判定された場合、ステップ747で、ポイントが旧世代から新世代につながっていることを示す新世代ポイントを設定する。次に、ステップ748で、ルート・アレイを割り当てて、割り当てられたルート・アレイを指し示すようにカードマークを設定する。処理は、ステップ748からステップ749に移り、ルート・アレイの割り当てに成功したか否かが判定される。ルート・アレイの割り当てが成功した場合には、ステップ750で、そのルート・アレイに追跡されたルートを挿入する。一方、ルート・アレイの割り当てが成功しなかった場合には、処理はステップ755に移り、そのルート・アレイがオーバーフローしていることを示すようにカードマークを設定する。例えば、ルート・アレイがオーバーフローしていることを示すようにカードマークを設定する処理は、カードマークに含まれるポイン

タ、即ち、27ビットのポイントを零に設定することにより実現される。

【0064】ステップ740でポイントが旧世代から新世代につながっていないと判定された場合、処理はステップ742に進み、旧世代内の下位のトレインを指し示すポイントが存在するか否かが判定される。ポイントが旧世代内の下位のトレインを指し示していると判定された場合、次に、ステップ760で、旧世代から新世代へのポイントが存在していることを示す新世代フラグが設定されているか否かの判定が行われる。新世代フラグが設定されている場合、処理はステップ775に進み、下位トレインフラグが設定されているか否かが判定される。実施例では、下位トレインフラグは、カードマークに含まれる最下位から三番目のビットであり、上述したように、ポイントが下位のトレインを指し示しているか否かを示している。下位トレインフラグが設定されていると判定された場合には、カードマークを更新する処理を完了する。一方、下位トレインフラグが設定されていないと判定された場合には、ステップ776で、下位トレインフラグを設定する。下位トレインフラグの設定後、カードマークの更新処理を完了する。

【0065】一方、ステップ760で新世代フラグが設定されていないと判定された場合には、処理はステップ766に進み、下位トレインフラグが設定されているか否かが判定される。下位トレインフラグが設定されていないとステップ766で判定されると、ステップ767で、下位トレインに対するポイントの存在を示す下位トレインフラグを設定する。下位トレインフラグが設定されると、次のステップ768で、ルート・アレイを割り当て、割り当てられたルート・アレイを指し示すようにカードマークを設定する。次に、ステップ769で、新しいルート・アレイの割り当てが成功したか否かが判定される。割り当てが成功した場合には、次に、ステップ770で、そのルート・アレイにまだ追跡してきたルートが存在していない場合には、ルート・アレイに追跡ルートを挿入する。その後、カードマークを更新する処理を完了する。一方、ステップ769で、新しいルート・アレイの割り当てに成功しなければ、処理がステップ769からステップ762に移り、ルート・アレイのオーバーフローを示すようにカードマークが設定される。ルート・アレイのオーバーフローを示すようにカードマークが設定された後、カードマークを更新する処理を完了する。

【0066】ステップ766で、下位トレインに対するポイントの存在を示す下位トレインフラグが設定されていると判定された場合、ステップ772で、対応するルート・アレイがオーバーフローしているか否かの判定が行われる。ルート・アレイがオーバーフローしている場合には、ステップ762で、ルート・アレイのオーバーフローを示すようにカードマークが設定される。一方、

ステップ772でルート・アレイがオーバーフローしていないと判定された場合には、ステップ770で、そのルート・アレイにまだ追跡してきたルートが存在していない場合には、ルート・アレイに追跡ルートを挿入する。

【0067】ステップ742で、ポインタが旧世代内の下位トレインを指し示していないと判定された場合、ステップ744で、同一トレイン内の下位のカーをポインタが指し示しているか否かが判定される。ポインタが同一トレイン内の下位のカーを指し示していない場合には、カードマークの更新処理を完了する。一方、ポインタが同一トレイン内の下位のカーを指し示していると判定された場合には、処理がステップ780に進み、新世代に対するポインタの存在を示す新世代フラグが設定されているか否かが判定される。

【0068】ステップ780で新世代フラグが設定されていると判定された場合には、次に、ステップ782で、同一トレインフラグが設定されているか否かの判定が行われる。即ち、同一トレイン内の下位のカーに対するポインタが存在しているか否かの判定が行われる。同一トレインフラグが設定されている場合には、カードマークを更新する処理を完了する。逆に、同一トレインフラグが設定されていない場合には、次のステップ783で同一トレインフラグを設定する。同一トレインフラグを設定した後、カードマークを更新する処理を完了する。

【0069】一方、ステップ780で新世代フラグが設定されていないと判定された場合には、処理がステップ786に進み、下位トレインに対するポインタの存在を示すカードマークの下位トレインフラグが設定されているか否かが判定される。下位トレインフラグが設定されている場合には、処理がステップ782に進み、同一トレインフラグが設定されているか否かの判定を行う。

【0070】下位トレインフラグが設定されていない場合、即ち、下位トレインに対するポインタが存在していない場合には、処理がステップ786からステップ788に移り、そのカードマークの同一トレインフラグが設定されているか否かが判定される。言い換えると、ステップ788で、そのカードマークが、同一トレインの下位のカードを示すポインタの存在を示しているか否かの判定が行われる。同一トレインフラグが設定されていると判定された場合には、次に、ステップ790で、そのカードマークに関するルート・アレイがオーバーフローしているか否かが判定される。ルート・アレイがオーバーフローしている場合には、ステップ793で、そのルート・アレイがオーバーフローしていることを示すようにカードマークが設定される。逆に、ルート・アレイがオーバーフローしていないと判定された場合には、次に、ステップ792で、そのルート・アレイに追跡してきたルートがまだ存在していない場合には、ルート・ア

レイに追跡ルートを挿入する。必要に応じて、ブロック・アレイにルートを挿入した後、カードマークを更新する処理を完了する。

【0071】一方、ステップ788で、同一トレインフラグが設定されていないと判定されると、次に、ステップ794で、ルート・アレイを割り当て、更に、割り当てられたルートアレイを指し示すようにカードマークを設定する。次のステップ796で、ルート・アレイの割り当てが成功したか否かの判定が行われる。ルート・アレイの割り当てが成功しなかった場合、ステップ782で、ルート・アレイのオーバーフローを示すようにカードマークが設定される。一方、ルート・アレイの割り当てが成功した場合には、処理はステップ792に進み、そのルート・アレイに追跡してきたルートがまだ存在していない場合には、ルート・アレイに追跡ルートを挿入する。

【0072】図13及び図14は、本発明の一実施例にしたがって、旧世代メモリ内で実行されるガーベジコレクション処理に関するステップ、即ち、図7のステップ608の詳細を示すフローチャートである。まず、ステップ802で、固定ルート、すなわち、本実施例では、旧世代メモリを直接指し示すルートを追跡する。本発明の一実施例にしたがって固定ルートを追跡する処理に関するステップは、図9ないし図12を参照して、上述した。

【0073】次に、ステップ804で、新世代からのルートを追跡する。即ち、新世代から旧世代につながるルートの関係を追跡する。次に、処理がステップ804からステップ806に進み、設定された下位トレインフラグを有する次の入力的位置決めをするために、カード・アレイ、即ち、カードマークのアレイを走査する。次のステップ808で、該当する入力、すなわち、設定された下位トレインフラグを有する入力が見つかったか否かが判定される。該当する入力が見つかったと判定された場合には、ステップ812で、見つかった入力に関して、新世代フラグが設定されているか、および／あるいは、ルート・アレイがオーバーフローしているかの判定が行われる。新世代フラグが設定されているか、および／あるいは、ルート・アレイがオーバーフローしている場合には、ステップ814で、カード内のすべてのオブジェクトを走査して、ルートを検索し、見つかったルートの追跡を行う。先にも説明したように、ルートの追跡に関するステップは、図9ないし図12を参照して上述した。ルートの追跡を実行後、処理はステップ806に戻り、設定された下位トレインフラグを有する次の入力を位置決めするように、カード・アレイの走査を行う。

【0074】ステップ808で見つけられた入力、即ち、カードマークに関して、新世代フラグも設定されていないし、ルート・アレイもオーバーフローしていないとステップ812で判定された場合には、処理がステッ

ブ816に移り、入力に係するルート・アレイを走査して、そのルート・アレイに係するルートを追跡する。ルートを追跡した後、ステップ806で、設定された下位トレインフラグを有する次の入力を検索するためにカード・アレイを走査する。

【0075】一方、ステップ808で、設定された下位トレインフラグを有する入力を探すためにカード・アレイを走査した結果、該当する入力が見つからなかった場合には、ステップ820で、すべての複製されたオブジェクトの走査が完了するまで、旧世代内でルートの追跡が実行される。例えば、他のトレインから参照されている所定のトレイン内のオブジェクトをそれらのトレインに複製する。すべての複製されたオブジェクトの走査が完了したら、ステップ830で、走査処理の間に、最下位トレインを指し示すルートに遭遇したか否かの判定が行われる。最下位トレインに対するルートに遭遇しなかったと判定された場合には、ステップ840で、最下位トレイン全体を解放する。言い換えると、最下位トレインによって保持されている旧世代メモリを解放する。最下位トレインに対するポインタが存在しない場合、最下位トレインはガーベジのみを含んでいるという事実に基づいて、最下位トレイン全体を解放するものでもよい。最下位トレイン全体を解放した場合、互いに参照しあい、同一トレイン内の異なったカーに位置する循環構造体を回収することができる。最下位トレイン全体を解放した後、旧世代ガーベジコレクション処理を終了する。

【0076】一方、最下位トレインへのルートに遭遇したとステップ830で判定されると、次に、ステップ832で、カード・アレイを走査して、設定された同一トレインフラグを有する次の入力、即ち、カードマークを探す。次のステップ834で、設定された同一トレインフラグを有する入力が見つかったか否かが判定される。ここで、設定された同一トレインフラグを有する入力が見つからなかったと判定された場合には、ステップ836で、すべての複製されたオブジェクトの走査が完了するまで、旧世代内部でルートの追跡が行われる。すべての複製されたオブジェクトを走査するとは、例えば、ポインタの推移的な検索を行うことである。すべての複製されたオブジェクトの走査が完了すると、ステップ837で、最下位トレイン内の最下位カーを解放して、その後、旧世代ガーベジコレクション処理を終了する。

【0077】逆に、ステップ834で、設定された同一トレインフラグを有する入力、即ち、カードマークが見つかったと判定された場合には、処理はステップ838に移り、その入力に関して、新世代フラグが設定されているか、下位トレインフラグが設定されているか、あるいは、ルート・アレイがオーバーフローしているかが判定される。入力が設定された新世代フラグのどれかまたはすべてを持つ場合には、処理がステップ850に進み、カード内のすべてのオブジェクトを走査して、ル

トを検索し、見つかったルートを追跡する。ルートの追跡後、処理はステップ832に戻り、設定された同一トレインフラグを有する次の入力を検索するために、カード・アレイの走査がもう一度行われる。

【0078】一方、ステップ834で見つかった入力に関して、新世代フラグも設定されていないし、下位トレインフラグも設定されていないし、また、ルート・アレイもオーバーフローしていないとステップ838で判定された場合には、ステップ856で、入力に係するルート・アレイが走査され、そのルート・アレイ内のルートの追跡が行われる。ルートの追跡後、処理はステップ832に戻り、設定された同一トレインフラグを有する次の入力を検索するために、カード・アレイの走査が行われる。

【0079】本発明にしたがって世代間ガーベジコレクションを利用するコンピュータプログラムは、種々の異なったコンピュータシステム上で実行可能である。図15に、本発明を実現するのに適した典型的な汎用コンピュータシステムを示す。コンピュータシステム930は、任意の数のプロセッサ932（中央処理装置即ちCPUと称する）と、プロセッサに接続される一次記憶装置934（通常、リードオンリーメモリ即ちROM）及び一次記憶装置936（通常、ランダムアクセスメモリ即ちRAM）とを含む記憶装置と、を備える。当業者に周知のように、ROMは、CPU932に対して、一方方向にデータおよび命令を伝送するのに対して、RAMは、双方向にデータおよび命令を伝送する。一次記憶装置934および936が、任意の適当なコンピュータ読み取り可能な媒体を備えるようにしてもよい。大容量記憶装置のような二次記憶媒体938も、CPU932に対して双方向に連結され、追加のデータ記憶容量を実現している。大容量記憶装置938は、コンピュータコード、データ等を含むプログラムを格納するために利用可能なコンピュータ読み取り可能な媒体であり、通常は、一次記憶装置934および936よりも処理速度の遅いハードディスクあるいはテープのような記憶媒体で構成されている。また、大容量記憶装置938を、磁気テープあるいは紙テープリーダや他の周知の装置の形態で実現するものとしてもよい。大容量記憶装置938内部に保持される情報を、仮想記憶のようにRAM936の一部として組み込むこともできる。CD-ROM等の所定の一次記憶装置934も、また、CPUに対して一方方向にデータを伝送することができる。

【0080】CPU932は、さらに、一つあるいは複数の入出力装置940に接続される。入出力装置940の例としては、これらに限定されるものではないが、ビデオモニター、トラックボール、マウス、キーボード、マイク、タッチ式ディスプレイ、変換器カードリーダ、磁気テープあるいは紙テープリーダ、タブレット、スタイラス、音声あるいは手書き認識装置、他のコンピュー

タ等の周知の入力装置が挙げられる。また、CPU932を、912で一般的に示されるようなネットワーク接続を用いて、インターネットやイントラネット等のコンピュータネットワークや遠距離通信ネットワークに任意に接続するようにしてもよい。このようなネットワーク接続を利用して、CPU932は、ネットワークから情報を入力することもできるし、また、前述した方法ステップを実行中にネットワークに情報を出力することも可能である。CPU932によって実行される一連の命令として示される、このような情報を、搬送波内に具現化されるコンピュータデータ信号の形態でネットワークから入力したり、ネットワークに出力したりすることができる。上述した装置や材料は、コンピュータハードウェア並びにソフトウェアの当業者には周知のものである。

【0081】以上、本発明の実施例をいくつか説明したが、本発明は、その要旨や範囲を逸脱することなく、様々な形態で具現可能である。例えば、新世代ガーベジコレクションや旧世代ガーベジコレクションを行う場合のステップの順番は変更可能である。また、本発明の要旨や範囲を逸脱することなく、いくつかのステップを除いたり、追加したりすることも可能である。

【0082】上述した実施例では、世代ガーベジコレクションをメモリの新世代および旧世代で実行されるものとして説明したが、世代ガーベジコレクションをメモリの多くの世代にまたがって実行するものとしてもよい。即ち、本発明の世代ガーベジコレクションの方法を、例えば、新世代、「中間世代」、旧世代等のように、各世代内のオブジェクトの年齢によって区別される複数の世代に分割可能なメモリ上で実行するようにしてもよい。複数の世代が存在する場合には、世代間ポインタは、通常、いずれか二つの世代をつなぐものである。

【0083】また、上述した実施例では、カードマークを、27ビットのポインタと、予備ビットと、同一トレインフラグと、下位トレインフラグと、新世代フラグと、汚染フラグとを含む32ビットのワードとして説明したが、本発明の要旨や範囲を逸脱することなく、カードマークを様々に変更可能である。例えば、あるシステムの所定の要件の少なくとも一部にしたがって、カードマークを32ビットより少ないビットで構成しても、あるいは、32ビットより多いビットで構成することもできる。カードマークを構成するビットは様々に変更可能である。例えば、カードマークを構成する32ビットが、26ビットのポインタと、2ビットの「ステータス・インジケータ（状態表示部）」あるいはタグビットを含むように構成してもよい。ステータス・インジケータビットの様々な組み合わせの値が、カードマークが汚染されているか、あるいは、クリーンか、また、カードマークが新世代ポインタを含むか、あるいは、含まないか、を示すように、ステータス・インジケータビットを構成するようにしてもよい。また、カードマークが、下

位トレインに対する何らかのポインタが存在するか否かを示す下位トレインフラグ、所定のトレイン内の下位のカーに対する何らかのポインタがそのトレイン内に存在するか否かを示す同一トレインフラグ、下位トレインオーバーフローフラグ、および、同一トレインオーバーフローフラグを含むように構成してもよい。

【0084】さらに、上述の実施例では、カードマーク内部の最下位ビットを、例えば、カードマークが汚染されているか否かを示すために用いられるフラグとして説明したが、カードマークのフラグは、カードマーク内の任意の適当な位置にあればよい。例えば、フラグを表わすビットが、カードマーク内部の最上位ビットでもよい。あるいは、フラグを表わすビットが、カードマークの中に点在していてもよい。

【0085】上述の実施例では、旧世代メモリ内のカーの大きさを固定したものとして説明したが、カーの大きさを動的に割り当てるようにしてもよい。即ち、あるカー内に位置するべきオブジェクトの所定の要件に基づいて、そのカーの大きさを決めるようにしてもよい。例えば、本発明の要旨や範囲から逸脱することなく、カー内部のオブジェクトに起因するポインタの数を最小にするように、ガーベジコレクション処理の実行中にカーの大きさを割り当てるようにしてもよい。

【0086】従来のガーベジコレクションの方法も旧世代メモリ内で状況に応じて実現可能である。例えば、旧世代メモリがいっぱいになった場合を考える。旧世代メモリがいっぱいの場合、カー上で旧世代ガーベジコレクションを実行しても、すぐに利用可能なように充分な量のメモリを解放することができない場合がある。このような時に、本発明の要旨や範囲から逸脱することなく、旧世代メモリがいっぱいの場合に、可能な限りたくさん量の旧世代メモリを解放するように、マーク掃引コレクション処理のような従来のガーベジコレクション処理を実行してもよい。このように、上述の実施例は例示に過ぎず、何ら発明を限定するものではない。本発明は、上述の説明に何ら限定されるものではなく、特許請求の範囲内で様々に変更可能である。

【図面の簡単な説明】

【図1】従来技術にしたがって、オブジェクトを含むコンピュータメモリの領域を例示する図である。

【図2】従来技術にしたがって、新世代と旧世代とに区分されるメモリとルートとの間のインターフェースを例示する図である。

【図3】従来技術にしたがって、記憶集合に関係するメモリを例示する図である。

【図4】本発明の一実施例にしたがって、トレインに分割される旧世代メモリを例示する図である。

【図5】本発明の一実施例にしたがって、カードマーク・アレイと関連するルート・アレイのプールとを例示する図である。

31

【図 6】本発明の一実施例にしたがって、コンピュータプログラムを実行するためのステップを示すフローチャートである。

【図 7】本発明の一実施例にしたがって、ガーベジコレクションを実行する処理、即ち、図 6 のステップ 510 の詳細を示すフローチャートである。

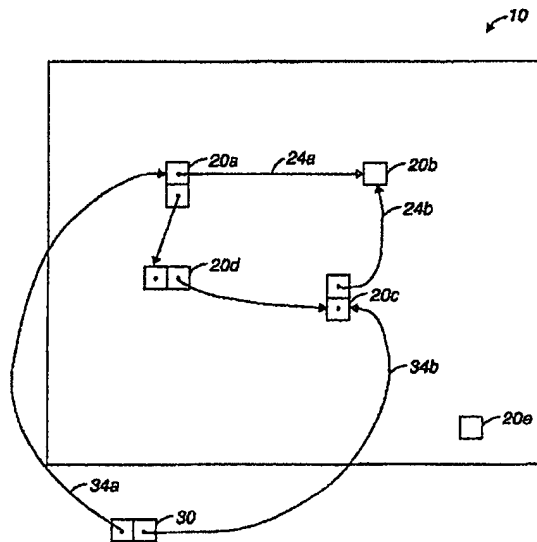
【図 8】本発明の一実施例にしたがって、新世代メモリ内で実行されるガーベジコレクション処理に関するステップ、即ち、図 7 のステップ 604 の詳細を示すフローチャートである。

【図 9】本発明の一実施例にしたがって、一つのルートを追跡する処理に関するステップを示すフローチャートである。

【図 10】本発明の一実施例にしたがって、カードマークを更新する処理に関するステップ、即ち、図 9 のステップ 728 の詳細を示すフローチャートである。

【図 11】本発明の一実施例にしたがって、カードマークを更新する処理に関するステップ、即ち、図 9 のステップ 728 の詳細を示すフローチャートである。 *

【図 1】



32

* 【図 12】本発明の一実施例にしたがって、カードマークを更新する処理に関するステップ、即ち、図 9 のステップ 728 の詳細を示すフローチャートである。

【図 13】本発明の一実施例にしたがって、旧世代ガーベジコレクション処理に関するステップ、即ち、図 7 のステップ 608 の詳細を示すフローチャートである。

【図 14】本発明の一実施例にしたがって、旧世代ガーベジコレクション処理に関するステップ、即ち、図 7 のステップ 608 の詳細を示すフローチャートである。

10 【図 15】本発明を実現するのに適したコンピュータシステムを例示する図である。

【符号の説明】

930…コンピュータシステム

932…CPU

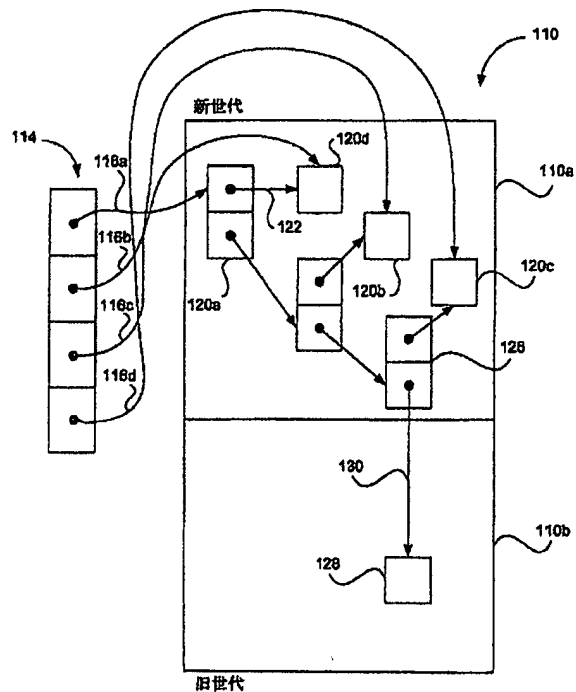
934…一次記憶装置

936…RAM

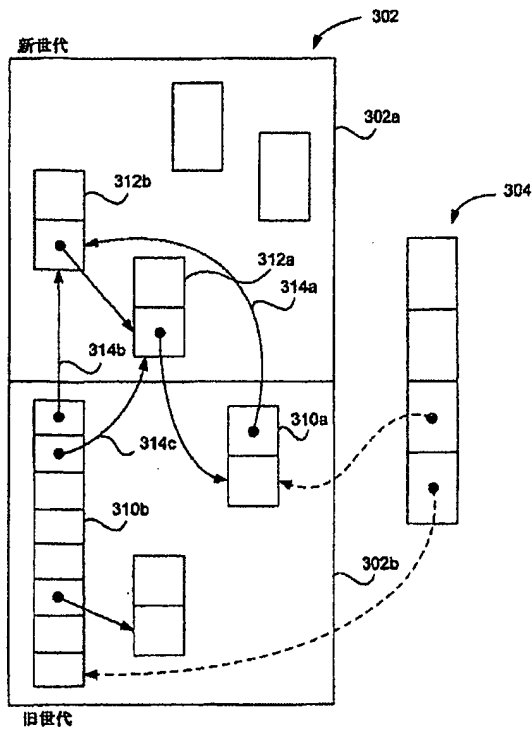
938…二次記憶媒体

940…入出力装置

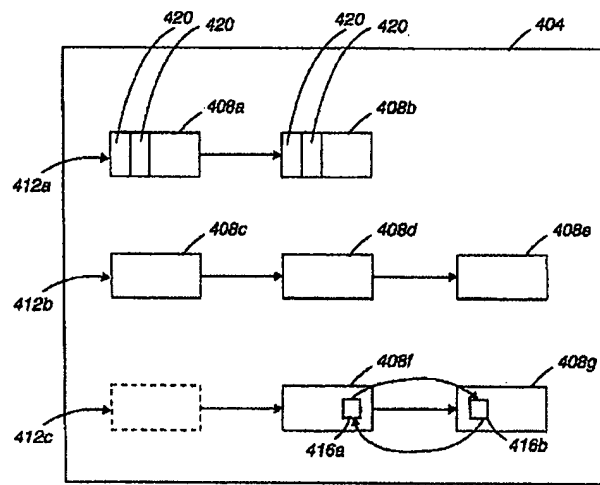
【図 2】



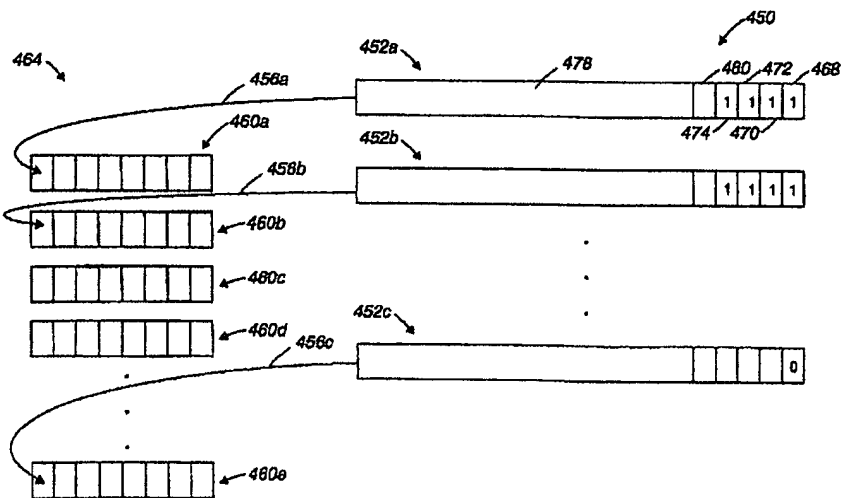
【図3】



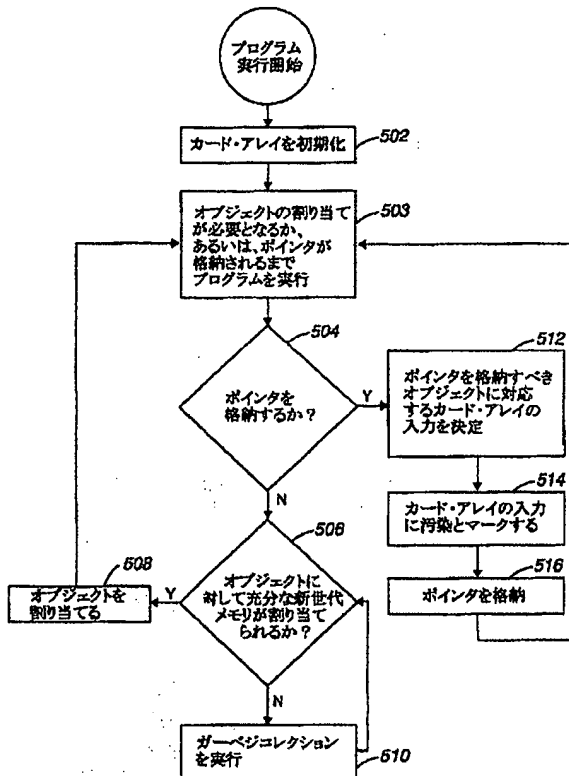
【図4】



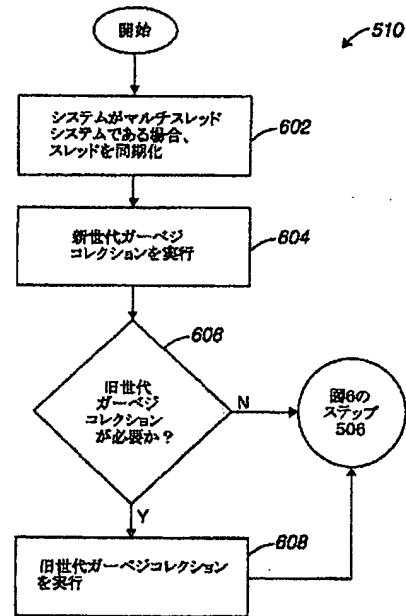
【図5】



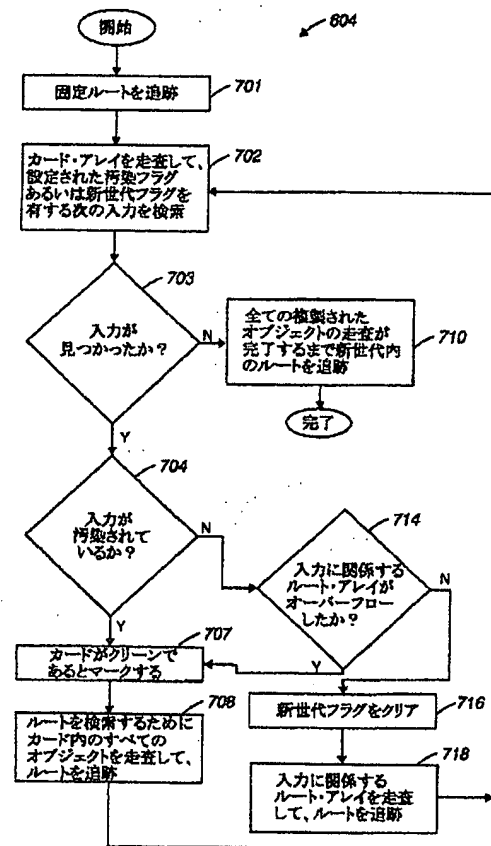
【図6】



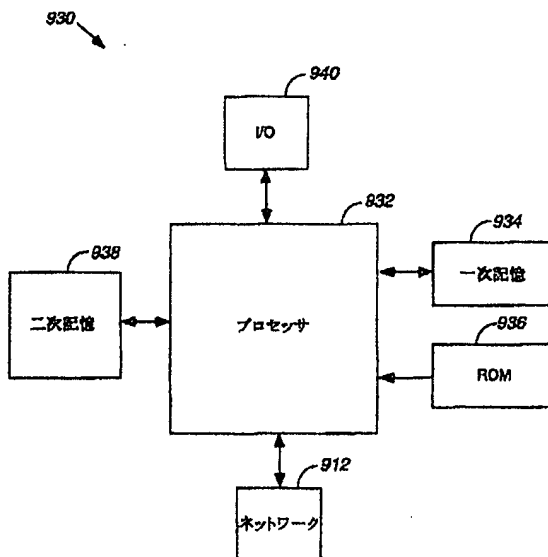
【図7】



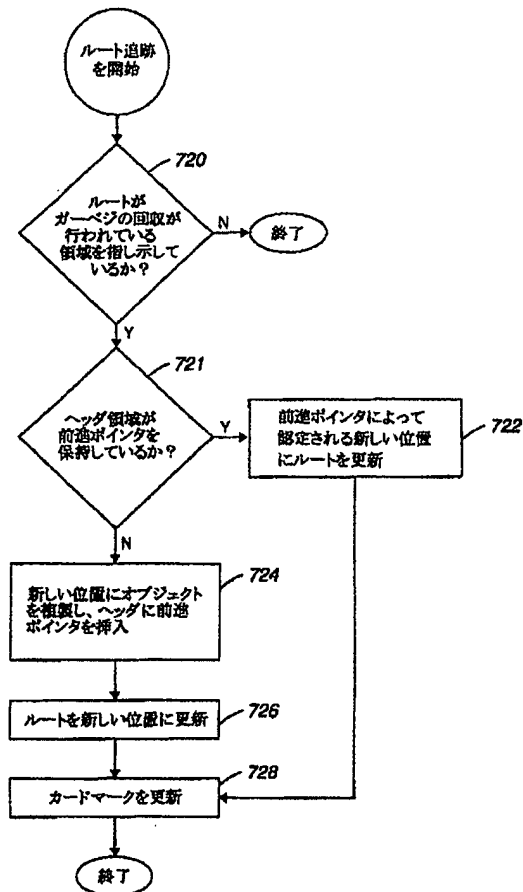
【図8】



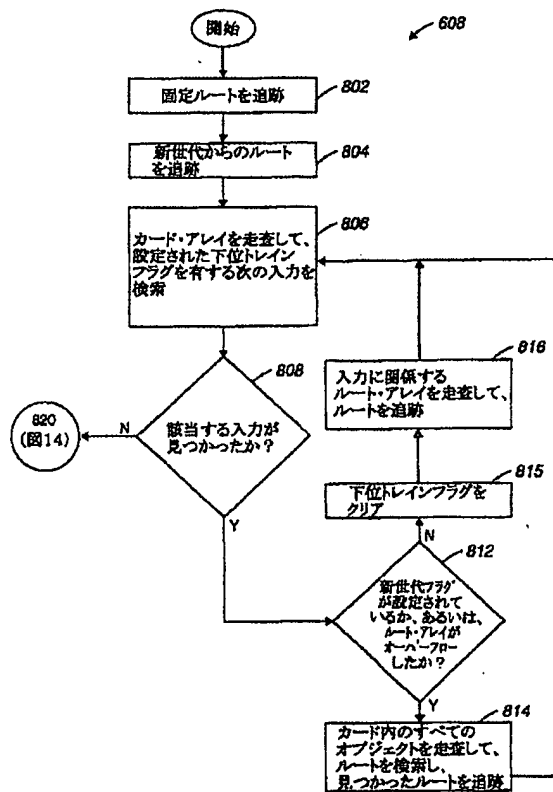
【図15】



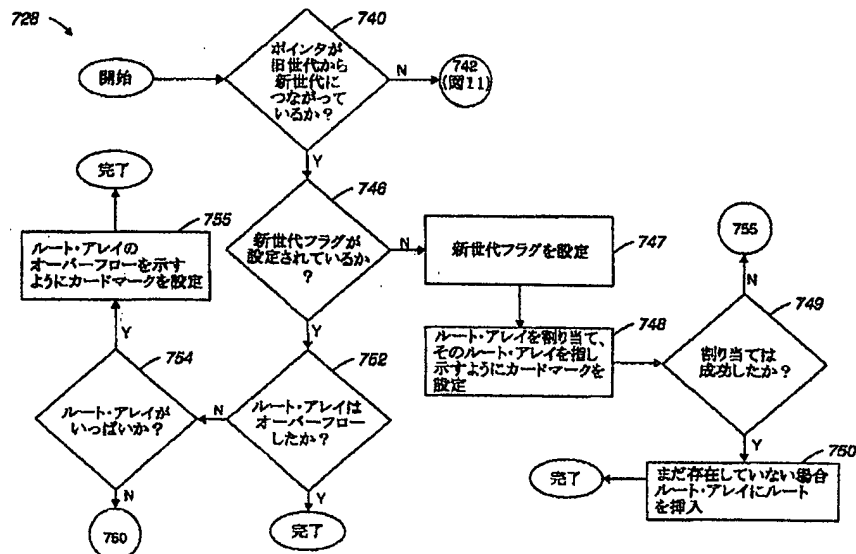
【図9】



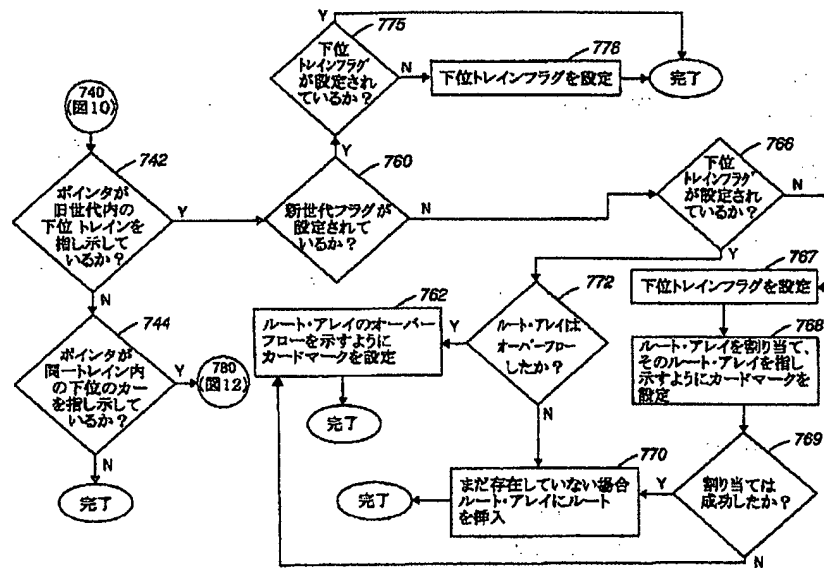
【図13】



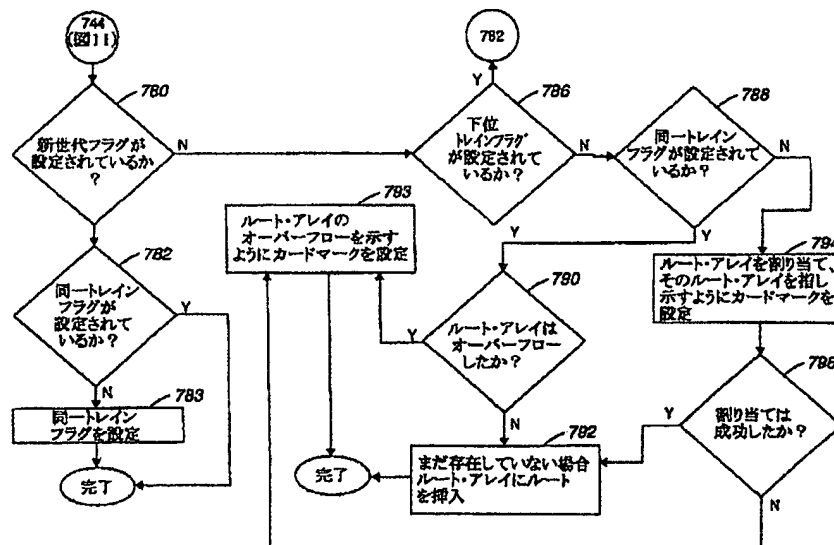
【図10】



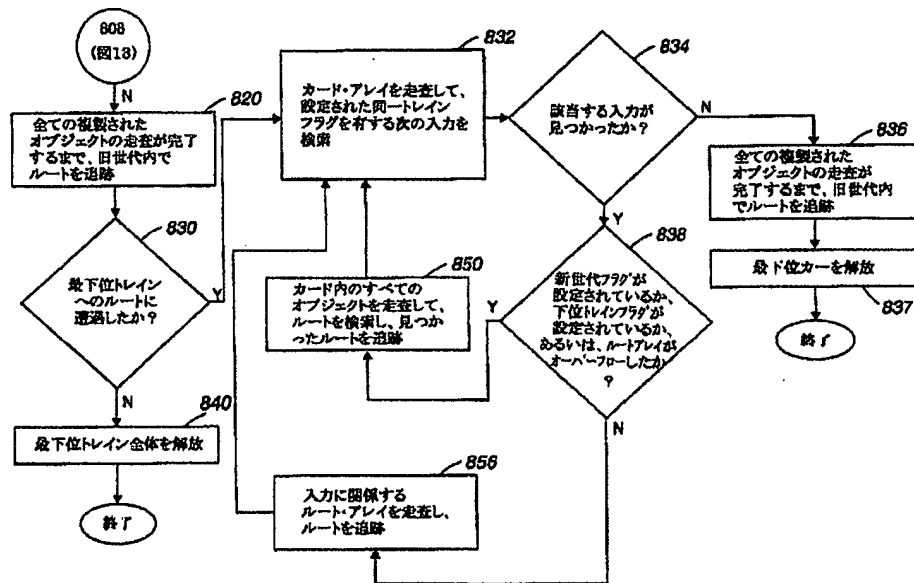
【図11】



【図12】



【図14】



フロントページの続き

(71)出願人 591064003

901 SAN ANTONIO ROAD
PALO ALTO, CA 94303, U.
S. A.

(72)発明者 ラース・バク

アメリカ合衆国 カリフォルニア州94303
パロ・アルト, コリーナ・ウェイ, 3782

【外国語明細書】

1 Title of Invention

Methods and Apparatus for Generational Dynamic Management of Computer Memory

2 Claims

1. A computer-implemented method for dynamically managing memory associated with a computer system, the memory including a first memory section and a second memory section that is divided into a plurality of blocks, the blocks in the second memory section each having an associated marker, the method comprising:

performing a first garbage collection on the first memory section;

performing a second garbage collection on a selected one of the blocks in the second memory section;

performing a third garbage collection on the selected block in the second memory section, wherein the third garbage collection includes determining whether the selected block includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the selected block, wherein the status includes an indication of whether the reference to the second object was stored after the second garbage collection was performed; and

when the reference to the second object was stored after the second garbage collection was performed, creating a new root array using the selected marker.

2. A computer-implemented method as recited in claim 1 wherein when it is determined that the first object does not reference a second object which is not included in the selected block, the method further includes releasing the selected block.

整理番号=PA52C757

ページ (2)

3. A computer-implemented method as recited in one of claims 1 and 2 wherein when it is determined that the selected block includes a first object which references a second object, the step of performing the third garbage collection further includes:

determining whether the second object is in the first memory section using the selected block marker; and

when it is determined that the second object is in the first memory section, a root array associated with the selected block marker is updated to insure that the root array associated with the selected block marker includes a pointer to the second object.

4. A computer-implemented method as recited in one of claims 1 and 2 wherein when it is determined that the selected block includes a first object which references a second object, the step of performing the third garbage collection further includes:

determining whether the second object is in the second memory section using the selected block marker; and

when it is determined that the second object is in the second memory section, a root array associated with the selected block marker is scanned to identify references to other objects.

5. A computer-implemented method as recited in claim 4 further including following the references to other objects.

6. A computer-implemented method as recited in one of claims 1 and 2 wherein the plurality of blocks are arranged as a plurality of trains, the selected block being included in a first one of the plurality of trains, the step of performing the third garbage collection further including:

整理番号=PA52C757

ページ (3)

determining whether the second object is in a second one of the plurality of trains based at least in part on the status indicated by the marker associated with the block.

7. A computer-implemented method as recited in claim 6 wherein when it is determined that the second object is not in a second one of the plurality of trains, the method further includes releasing the second train.

8. A computer-implemented method as recited in one of claims 1 and 2 wherein the plurality of blocks are arranged as a plurality of cars, the plurality of cars further being arranged as a plurality of trains, the selected block being included in a first one of the plurality of cars that forms a selected one of the plurality of trains, the step of performing the third garbage collection further including:

determining whether the second object is in a second one of the plurality of cars that forms the selected train based at least in part on the status indicated by the marker associated with the block.

9. A computer-implemented method as recited in claim 8 wherein when it is determined that the second object is not in the second car of the selected train, the method further includes releasing the second car.

10. A computer-implemented method for dynamically managing memory associated with a computer system, the memory being divided into a plurality of blocks, the blocks each having an associated marker, the method comprising:

performing a first garbage collection on a selected one of the blocks;

performing a second garbage collection on the selected block, wherein the second garbage collection includes determining whether the selected block includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the

整理番号=PA52C757

ページ (4)

selected block, the status indicated by the marker including an indication of whether the reference to the second object was stored after the first garbage collection was performed; and

when the reference to the second object was stored after the first garbage collection was performed, creating a new root array using the selected block.

11. A computer-implemented method as recited in claim 10 wherein when it is determined that the first object does not reference a second object which is not included in the selected block, the method further includes releasing the selected block.

12. A computer-implemented method as recited in one of claims 10 and 11 wherein the plurality of blocks are arranged as a plurality of trains, the selected block being included in a first one of the plurality of trains, the step of performing the garbage collection further including:

determining whether the second object is in a second one of the plurality of trains based at least in part on the status indicated by the marker associated with the block; and

when it is determined that the second object is not in a second one of the plurality of trains, the method further includes releasing the second train.

13. A computer-implemented method as recited in claim 10 further including an additional memory section, wherein when it is determined that the second object is not included in the selected block, the step of performing the garbage collection on the selected block further includes determining whether the second object is included in the additional memory section based at least in part on the status indicated by the marker associated with the selected block.

14. A computer-implemented method for allocating a selected object in computer memory, the computer memory including a first section, a second section,

整理番号=PA52C757

ページ (5)

and an array of words associated with elements in the second section, the words being arranged to indicate whether the contents of their associated elements reference objects in the first section, the method comprising:

determining whether the first section is suitable for the selected object to be allocated;

allocating the selected object when it is determined that the first section is suitable for the object to be allocated; and

performing a garbage collection in the first section when it is determined that the first section is not suitable for the selected object to be allocated, wherein performing the garbage collection includes scanning the array of words to locate selected words that indicate that their associated elements in the second section reference particular objects in the first section, and wherein the particular objects in the first section are retained during the garbage collection.

15. A computer-implemented method as recited in claim 14 wherein when the selected markers that indicate that their associated elements in the second section reference the particular objects are located, performing the garbage collection in the first section further includes:

following associations of roots within the associated elements in the second section, wherein following the associations of the roots includes updating the roots to new locations within the computer memory and updating the selected words.

16. A computer-implemented method as recited in claim 15 wherein updating the selected words includes:

determining when a selected one of the roots is in the second section and references a selected one of the particular objects in the first section; and

when it is determined that the selected root is in the second section and references the selected particular object in the first section, updating the selected

整理番号=PA52C757

ページ (6)

words to indicate that the selected root is in the second section and the selected particular object is in the first section.

17. A computer-implemented method as recited in claim 14 further including:
performing a garbage collection in the second section, wherein performing the garbage collection includes following associations of roots associated with selected ones of the elements, the roots being associated with the new generation, whereby following the associations of the roots involves updating some of the words in the array of words.

18. A marker for use in a computer system including a first memory section and a second memory section that is arranged as a plurality of elements, the marker being associated with a first element selected from the plurality of elements, the marker comprising:

a first indicator arranged to identify whether the first selected element references an object contained within the first memory section.

19. A marker for use in a computer system as recited in claim 18 further including a reference to the object, the reference to the object being a pointer to an array which contains an address for the object.

20. A computer system comprising:
memory including a first memory section and a second memory section that is segmented into a plurality of blocks;

an array of markers, each marker corresponding to an associated block in the second memory section, the markers being configured to indicate when their associated blocks reference objects outside of their associated block;

a set of root arrays, the root arrays being arranged to hold addresses of the objects, wherein selected markers from the array of markers are associated with the set of root arrays;

整理番号=PA52C757

ページ (7)

a first garbage collector for reclaiming space within the first memory section; and

a second garbage collector for reclaiming space within the second memory section based at least in part upon the status of the markers in the array of markers.

21. A computer system as recited in claim 20 wherein a first marker selected from the array of markers includes a section indicator that identifies when the associated block of the first marker references a first object in the first memory section.

22. A computer system as recited in one of claims 20 and 21 wherein the plurality of blocks is arranged as a plurality of cars, the plurality of cars further being arranged as a plurality of trains.

23. A computer system comprising:
memory that is segmented into a plurality of blocks;
an array of markers, each marker corresponding to an associated block in the memory, the markers being configured to indicate when their associated blocks reference objects outside of their associated block;

a set of root arrays, the root arrays being arranged to hold addresses of the objects, wherein selected markers from the array of markers are associated with the set of root arrays; and

a garbage collector for reclaiming space within the memory based at least in part upon the status of the markers in the array of markers.

24. A computer program product for dynamically managing memory associated with a computer system, the computer program product comprising:

computer code that performs a first garbage collection in a first memory section of the computer;

整理番号 = PA52C757

ページ (8)

computer code that performs a second garbage collection on a second memory section of the computer, the second memory section being divided into a plurality of blocks, wherein the second garbage collection includes determining whether a block selected from the plurality of blocks includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the selected block; and

a computer readable medium that stores the computer codes.

25. A computer program product for causing a processor of a computer system to dynamically manage memory associated with the computer system, the computer program product comprising:

computer code that performs a first garbage collection in a first memory section of the computer;

computer code that performs a second garbage collection on a second memory section of the computer, the second memory section being divided into a plurality of blocks, wherein the second garbage collection includes determining whether a block selected from the plurality of blocks includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the selected block; and

a computer data signal embodied in a carrier wave that represents the computer codes.

3 Detailed Description of Invention

1. Field of Invention

The invention relates generally to the management of dynamically allocated memory in a computer system. More particularly, the invention relates to tracking references between separate sections of memory associated with a computer system such that automatic storage-reclamation may be performed locally.

整理番号=PA52C757

ページ (9)

2. Description of the Relevant Art

The amount of memory associated with a computer system is typically limited. As such, memory must generally be conserved and recycled. Many computer programming languages enable software developers to dynamically allocate memory within a computer system. Some programming languages require explicit manual deallocation of previously allocated memory, which may be complicated and prone to error. Languages which require explicit manual memory management include the C and C++ programming languages. Other programming languages utilize automatic storage-reclamation to reclaim memory that is no longer necessary to ensure the proper operation of computer programs which allocate memory from the reclamation system. Such automatic storage-reclamation systems reclaim memory without explicit instructions or calls from computer programs which were previously utilizing the memory.

In object-oriented or object-based systems, the typical unit of memory allocation is commonly referred to as an object or a memory object, as will be appreciated by those skilled in the art. Objects which are in use are generally referred to as "live" objects, whereas objects which are no longer needed to correctly execute computer programs are typically referred to as "garbage" objects. The act of reclaiming garbage objects is commonly referred to as garbage collection, while an automatic storage-reclamation system is often referred to as a garbage collector. Computer programs which use automatic storage-reclamation systems are known as mutators due to the fact that such computer programs can change live memory objects during execution. Computer programs written in languages such as the Java™ programming language (developed by Sun Microsystems, Inc. of Palo Alto, California) and the Smalltalk programming language use garbage collection to automatically manage memory.

Objects typically contain references to other objects. As such, an area of computer memory which is managed by a garbage collector will generally contain a

整理番号=PA52C757

ページ (10)

set of objects which reference one another. Figure 1 is a diagrammatic representation of an area of computer memory which contains objects. A managed area of memory 10, which is typically a heap associated with a computer system, includes objects 20. In general, an object 20 may be referenced by other objects 20. By way of example, object 20a has a pointer 24a to object 20b. Object 20c also has a pointer 24b to object 20b. As such, object 20b is referenced by both objects 20a and 20c.

A number of external references into memory 10. As shown, a fixed root 30 which is external to memory 10 includes pointers 34 to objects, *e.g.*, objects 20a and 20c, located in memory 10. All objects, as for example objects 20a-d, which may be reachable by following references from fixed root 30 are considered to be live objects. Alternatively, object 20e, which is not reachable by following references from fixed root 30, is characterized as a garbage object.

Garbage collectors are typically implemented to identify garbage objects such as object 20e. In general, garbage collectors may operate using a number of different algorithms. Conventional garbage collection algorithms include reference counting collectors, mark sweep collectors, and copying collectors. As will be appreciated by those skilled in the art, during garbage collection, when objects 20 are moved, references to objects 20 must be adjusted accordingly.

It is often beneficial to separate a managed memory area into smaller sections to enable garbage collection to be performed locally in one area at a time. One memory partitioning scheme is generational garbage collection, in which objects are separated based upon their lifetimes as measured from the time the objects were created. "Younger" objects have been observed as being more likely to become garbage than "older" objects. As such, generational garbage collection may be used to increase the overall efficiency of memory reclamation.

整理番号=PA52C757

ページ (11)

Figure 2 is a diagrammatic representation of an interface between a root and memory which is partitioned into a new generation and an old generation. A memory 110, which is typically a heap that is associated with a computer system, includes a new generation 110a and an old generation 110b. A fixed root 114, or a global object which references objects within either or both new generation 110a and old generation 110b, includes pointers 116 to objects 120 in new generation 110a, as shown. Root 114 may be located on a stack, as will be appreciated by those skilled in the art.

Some objects 120 within new generation 110a, as for example object 120a, may also be considered as roots, since object 120a is assumed to be live and includes a pointer 122 to another object 120d. When new generation object 126 is live and points to old generation object 128, garbage collection performed in old generation 110b does not generally "collect" object 128. However, if new generation object 126 is dead, a garbage collection performed in new generation 110a will result in old generation object 128 becoming unreachable, since old generation object 128 will not be pointed to by any other object. If old generation object 128 is unreachable, then garbage collection performed in old generation 110b will result in the collection of old generation object 128. It should be appreciated that pointer 130, which points between new generation object 126 and old generation object 128 is considered to be an inter-generational pointer, since pointer 130 spans both new generation 110a and old generation 110b. When pointer 130 points from new generation object 126 to old generation object 128, old generation object 128 is considered to be tenured garbage, as old generation object 128 is not collectable using new generation garbage collection.

Another memory partitioning scheme involves separating memory into smaller areas in order to reduce the amount of time required for a single garbage collection to be performed. Pauses caused by garbage collection often tend to disrupt an associated mutator and are, therefore, undesirable. In some systems,

整理番号=PA52C757

ページ (12)

the garbage collector may be able to provide a guaranteed small maximum pause duration. Such garbage collectors associated are known as real-time garbage collectors. In other systems, the garbage collector may attempt to keep pause times small, but may fail to do so in some situations. Garbage collectors which attempt to keep pause times small are known as non-disruptive or incremental garbage collectors.

In order to operate on an individual memory area, a garbage collector must have knowledge of all references into that area. References into an area are referred to as roots for that area. It should be appreciated that roots may include both external references, *e.g.*, fixed roots, and references from other areas of computer memory. Accordingly, garbage collectors generally provide mechanisms for finding and tracking roots, or references.

One method of locating references into a memory area involves scanning through all objects in memory. For most systems, scanning through all objects in memory is prohibitively time-consuming. As such, a more elaborate tracking scheme is often required.

Whenever a mutator stores a reference into an object, additional processing may be implemented in order to track references for garbage collection purposes. This additional processing is known as a write barrier or store check. In order for efficiency of the mutator to be maintained at an acceptable level, the costs associated with the write barrier must be kept as low as possible.

One way of tracking references into a memory area involves maintaining a set which holds all roots for the area. Such a set is generally known as a remembered set for the area. The associated write barrier will detect when a reference into an area is stored. Accordingly, the write barrier will insert the location of the reference into the remembered set associated with the area.

整理番号=PA52C757

ページ (13)

Figure 3 is a diagrammatic representation of pointers between objects in a new generation and objects in an old generation which are tracked using a remembered set. A memory 302 is divided into a new generation 302a and an old generation 302b. A remembered set 304 is used to track pointers 314 which point from old generation objects 310 to new generation objects 312. The address of old generation object 310a is stored in remembered set 304, as old generation object 310a includes pointer 314a to new generation object 312b. Similarly, the address of old generation object 310b, which includes pointers 314b and 314c to new generation objects 312b and 312a, respectively, is also stored in remembered set 304.

Locating roots for garbage collection is straightforward when a remembered set is used, since the remembered set will contain all of the roots. However, the use of a write barrier is often expensive, as excess memory may be required. Also, when a new location is to be inserted into a remembered set, it is possible that the particular location may already be present within the remembered set. Checking the remembered set for duplicate locations prior to inserting a location is expensive. On the other hand, eliminating a check for duplicate locations may cause the remembered set to grow unnecessarily large. As will be appreciated by those skilled in the art, there is generally no upper bound on the number of duplicates which a remembered set may hold. When a reference to a location is to be stored, the location often already contains a previous reference. As a result, a remembered set entry that is associated with the location prior to a storage operation will generally become invalid following the storage operation. Removing an old entry may prove to be a costly operation in some situations, whereas leaving an old entry in place may result in the occupation of excess memory by a remembered set.

整理番号=PA52C757

ページ (14)

Another scheme which is often used for tracking references into a memory area is known as card marking. In general, card marking involves conceptually dividing memory into relatively small parts called cards. A garbage collector will then allocate an array of bits with one entry per card. When a reference is stored, the write barrier will compute the corresponding card array entry and set the associated bit. This process is known as "dirtying" the card. It should be appreciated that for efficiency reasons, a byte or a word array are often used in lieu of a bit array.

One advantage of using card marking is that the write barrier is relatively cheap. Another advantage of using card marking is that the amount of memory required for the card array is fixed. However, the processing required for locating roots at garbage collection time is often significant, *e.g.*, more processing is required than would be for a system which utilizes a remembered set. The garbage collector only knows approximately where reference stores have occurred. That is, the garbage collector only knows where the card marking array has dirty entries. Once a dirty entry is identified, the corresponding cards must be scanned for roots. Although scanning a card for roots is generally much cheaper than scanning an entire memory, scanning the card is still often costly. As will be understood by those skilled in the art, when a garbage collector locates a root, the corresponding card array entry must be kept dirty in order for the root to be located the next time garbage collection is invoked.

A combined scheme may use remembered sets and card marking. A write barrier will perform card marking as described above. However, at garbage collection time, the garbage collector will construct remembered sets for each card. When garbage collection is completed, the card marking array may then be cleared. Using a combined remembered set and card marking scheme reduces the amount of scanning required for subsequent garbage collections, thereby increasing the overall efficiency of garbage collection processes. However, the implementation of such

整理番号=PA52C757

ページ (15)

schemes is often complicated. Therefore, what is desired is a method and an apparatus for efficiently implementing a combined remembered set and card marking scheme.

SUMMARY OF THE INVENTION

The present invention relates to methods and apparatus for performing generational garbage collection within computer memory. According to one aspect of the present invention, a computer-implemented method for dynamically managing memory which includes a first memory section and a second memory section that is divided into a plurality of blocks each having an associated marker, includes performing a first garbage collection on the first memory section. The method also includes performing a second garbage collection on a selected one of the blocks in the second memory section. A third garbage collection is performed on the selected block in the second memory section. The third garbage collection includes determining whether the selected block includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the selected block. The status includes an indication of whether the reference to the second object was stored after the second garbage collection was performed, and if the reference to the second object was stored after the second garbage collection was performed, a new root array is created using the selected marker.

In accordance with another aspect of the present invention, a computer-implemented method for allocating a selected object in computer memory involves determining whether a first section of the memory is suitable for the selected object to be allocated, and allocating the selected object when it is determined that the first section is suitable for the object to be allocated. The method also involves performing a garbage collection in the first section when it is determined that the first section is not suitable for the selected object to be allocated. Performing the

整理番号=PA52C757

ページ (16)

garbage collection in the first section includes scanning an array of words to locate a selected word that is arranged to indicate that a first root associated with the word references an object which already exists in the memory.

In accordance with still another aspect of the present invention, a computer system includes a memory arranged as a first memory section and a second memory section, where the second memory section is divided into a plurality of blocks. Such a computer system also includes a processor that is coupled to the memory, and a first garbage collector which is associated with the first memory section. A second garbage collector associated with the second memory section is arranged to perform a garbage collection on a selected block which has an associated marker that is arranged to indicate whether the selected block includes a first object which references a second object that is not included in the selected block. The selected markers is associated with a root arrays which hold address of the object when the object references the second object that is not in the selected block. In one embodiment, the second garbage collector is arranged to use the marker to identify whether the second object is in the first memory section.

These and other advantages of the present invention will become apparent upon reading the following detailed descriptions and studying the various figures of the drawings.

DETAILED DESCRIPTION OF THE EMBODIMENTS

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings.

Garbage collection is used to dynamically allocate computer memory for use by a computer program. In general, garbage collection causes pauses in the execution of the program. One apparatus for reducing the pause-times associated

整理番号=PA52C757

ページ (17)

with garbage collection is to divide the memory into a plurality of generations. By way of example, in a two generation memory structure, the memory may be divided into a new generation, which includes newly allocated objects, and an old generation, which includes older objects. It should be appreciated that, newly allocated objects tend to die more quickly than older objects. As such, garbage collection may be performed more often in the new generation than in the old generation. Typically, the new generation is relatively small compared to the overall size of the memory. Accordingly, garbage collection within the new generation may typically be performed without causing substantial interruptions during the execution of a program.

While garbage collection in the new generation may not result in significant pause-times, garbage collection in the old generation may still cause significant pauses during the execution of a program, since the old generation is typically larger than the new generation. Further, if inter-generational pointers from objects in the old generation to objects in the new generation exist, then the inter-generational pointers are often tracked while a program executes. The inter-generational pointers are tracked since objects in the old generation that are associated with such pointers are considered to be roots of the new generation. The tracking of the pointers, or references, from objects in the old generation to objects in the new generation is often complicated and, therefore, time-consuming.

By dividing old generation memory into smaller areas, or blocks, pause-times associated with old generation garbage collection may be reduced. Old generation garbage collection may be performed in a specific block of old generation memory each time old generation garbage collection is performed. By limiting the blocks in which old generation garbage collection is performed, and by collecting garbage in different blocks each time old generation garbage collection is performed, the efficiency of the old generation garbage collection may be enhanced.

整理番号=PA52C757

ページ (18)

As previously mentioned, old generation memory is divided into blocks to facilitate old generation garbage collection. Each time old generation garbage collection is performed, the old generation garbage collection is performed on any suitable block. That is, old generation garbage collection may be performed incrementally. In one embodiment, garbage collection may be performed on the oldest block in which garbage collection has not been performed. In another embodiment, garbage collection may be performed on the block in which garbage collection has been performed least recently.

With reference to Figure 4a, the division of memory into blocks within an old generation will be described in accordance with an embodiment of the present invention. An old generation 404 includes blocks 408 of memory. The blocks of memory are typically allocated on an as-needed basis. That is, additional blocks are allocated when substantially all existing blocks 408 are full. Blocks 408, which contain objects associated with a particular program, are often referred to as "cars." Blocks or cars 408 are arranged in linked sets which are known as "trains" 412. For example, train 412a includes cars 408a, 408b, while train 412b includes cars 408c, 408d, 408e, and train 412c includes cars 408f and 408g.

Cars 408 within trains 412 are generally ordered based upon when a particular car 408 was added to a particular train 412. For example, within train 412a, if car 408b is added to train 412a after car 408a, car 408a is considered to be a lower car in the same train as car 408b. Similarly, trains 412 are generally also ordered based upon when a particular train 412 is created. As shown, if train 412b is added to memory 404 after train 412c, then train 412c is a lower train with respect to train 412b and, therefore, cars 408c, 408d, 408e, within train 412b.

An algorithm which uses trains 412 to perform garbage collection within old generation 404 typically collects a set number of cars 408 at a time. In the described embodiment, only a single car 408 is collected during each old generation

整理番号=PA52C757

ページ (19)

garbage collection, although it should be appreciated that any suitable number of cars may be collected during each old generation garbage collection. By collecting one car 408 at a time, the overall length of old generation garbage collection, and, therefore, the pause-times associated with old generation garbage collection is essentially bounded. Hence, the lengthy pause-times that are typically associated with old generation garbage collection may be reduced by collecting only a set number of cars 408 during each old generation garbage collection.

In general, in order to perform garbage collection in cars 408, all pointers which either originate in, or access objects within, each car 408 in old generation memory are tracked. One method for tracking pointers will be described below with respect to Figure 4b. When a particular car 408 which is to be collected includes a live object, the object is copied into a train 412 which either points to or is pointed to by the object. Then, once the copy of the object is made, the "original" object is collected or reclaimed. In general, when a selected train 412 no longer has pointers outside of itself, the selected train 412 may be eliminated, *i.e.*, the memory allocated to the selected train 412 is freed. Similarly, when a selected car 408 no longer has pointers outside of itself, the selected car 408 may be eliminated.

In the described embodiment, the size of cars 408 within a given train 412 is fixed. That is, all cars 408 may be of substantially of the same size. Therefore, since a single car 408 may not have enough space to accommodate linked objects, an object 416a in one car 408, *e.g.*, car 408f, may be linked to an object 416b in a second car 408, *e.g.*, car 408g. During an old generation garbage collection, objects 416a, 416b are likely to survive, since objects 416a, 416b are in separate cars 408, and reference each other. As such, an old generation garbage collection may not result in the collection of cars 408f, 408g. Hence, memory associated with cars 408f, 408g will not generally be freed unless pointers between cars 408f, 408g within train 412c are tracked.

整理番号=PA52C757

ページ (20)

In general, the length of trains 412, as well as the size of cars 408 may be widely varied. The size of cars 408 may depend on factors which include, but are not limited to, the size of old generation 404, the speed of the garbage collection algorithm, and the speed of the computer running the garbage collection algorithm. By way of example, car sizes on the order of approximately 20 kilobytes of memory to approximately 100 kilobytes of memory work well in current systems. In the described embodiment, cars 408 include approximately 64 kilobytes of memory.

Cars 408 may generally be divided into cards 420 which may be used to facilitate old generation garbage collection performed on cars 408, as will be described below with respect to Figure 4b. Cards 420, like cars 408, may take on any suitable size, although cards 420 may often include approximately 100 to approximately 1000 bytes. For example, for cars 408 which include approximately 64 kilobytes of memory, cards 420 may include approximately 512 bytes.

The efficiency of tracking inter-generational pointers which point from old generation objects to new generation objects may be improved by providing storage, or markers, that are associated with the different areas within the old generation. In one embodiment, a marker is a word that may include a flag which is set to indicate whether an associated portion of a particular area includes a pointer into an object in the new generation. The use of such words provides an efficient structure which may be used to track inter-generational pointers from objects in the old generation to objects in the new generation. Although a marker is described as being a word, it should be appreciated that a marker may be any suitable storage type, as for example bits, bytes, and the like.

In order to track inter-generational pointers from objects associated with cards in old generation memory to objects in new generation memory, cards may be provided with "tracking structures." The tracking structures, or card marks, may be used to identify whether inter-generational pointers exist for a card, and also

整理番号=PA52C757

ページ (21)

which objects within the card include the pointers. Figure 4b is a diagrammatic representation of an array of card marks and an associated pool of root arrays in accordance with an embodiment of the present invention. A root array is generally a small array which contains root locations which are specified as offsets relative to a card. In other words, a root array is essentially a small, remembered set. An array 450 of card marks 452 includes pointers 456 to root arrays 460 contained in a pool 464 of root arrays 460. Card marks 452 are generally associated with a car within a train. Specifically, a car within a train may be divided into a number of cards that hold old generation objects. As such, a car within a train may be associated with many card marks 452. A grouping of card marks 452, which may or may not include all card marks 452 associated with a car, is considered to be an array 450 of card marks 452. In the described embodiment, array 450 includes all card marks 452 associated with the old generation.

In the described embodiment, a card mark 452, as for example card mark 452a, is a 32-bit word, although it should be appreciated that card mark 452a may generally be a word of any suitable length. Alternatively, the data structure that constitutes the card mark may be divided into a number of components. Although the bits which make up card mark 452a may be assigned in any manner that is appropriate to a particular application, in one embodiment, the lowest bits, or least significant bits, of card mark 452a are used as flags which essentially identify the "status," or overall condition, of card mark 452a. A lowest bit 468 of card mark 452a is a "dirty" flag. That is, lowest bit 468 is generally arranged to indicate whether card mark 452a or, more specifically, a card (as shown in Figure 4a) associated with card mark 452a, is "dirty." Dirty cards are cards where pointers have been stored since the last garbage collection. Such pointers may include, but are not limited to, an inter-generational pointer which points from old generation memory into new generation memory, a pointer to a lower train, and a pointer to a lower car in the same train that is associated with card mark 452a. When lowest bit

整理番号=PA52C757

ページ (22)

468 is set, *i.e.*, when lowest bit 468 is set to "0," the indication is that card mark 452a is dirty.

A second lowest bit 470 of card mark 452a is a new generation flag, and is used to indicate whether card mark 452a includes an inter-generational pointer to a new generation. When second lowest bit 470 is set, *i.e.*, set to "0", the indication is that card mark 452a includes an inter-generational pointer to a new generation. A third lowest bit 472 is a lower train flag in the described embodiment. The lower train flag is arranged to indicate whether card mark 452a has a pointer into a lower train. When the lower train flag, or third lowest bit 472, is set, the indication is that card mark 452a includes a pointer into a lower train. A fourth lowest bit 474 is a same train flag that is arranged to indicate whether card mark 452a includes a pointer into a lower car in the same train.

In the described embodiment, in order for card mark 452a to be considered clean, card mark 452a generally may not include a pointer which points from an old generation to a new generation, a pointer into a lower train, or a pointer into a lower car of the same train. As such, lowest bit 468, second lowest bit 470, third lowest bit 472, and fourth lowest bit 474 all have values of "1" in the event that card mark 452a is clean. In other words, the lowest four bits of card mark 452a are cleared when card mark 452a is clean. Therefore, as shown, card marks 452a, 452b are clean, whereas card mark 452c, which has a lowest bit that is set, is dirty.

The remaining twenty-eight bits in card mark 452a may all be allocated for use as a pointer that points to an associated root array 460 which essentially holds addresses of pointers associated with card mark 452a. Alternatively, as for example in the described embodiment, card mark 452a includes a 27-bit pointer 478 and a reserved bit 480. 27-bit pointer 478 identifies pointer 456a which points from card mark 452a into root array 460a. Root array 460a is often either made up of eight words or sixteen words, although the number of words in root array 460a

整理番号=PA52C757

ページ (23)

may be widely varied. Root arrays 460 are generally arranged to hold references to roots. By way of example, root array 460a holds roots associated with card mark 450a and, hence, a card (as shown in Figure 4a) that is associated with card mark 450a.

When a suitable root array 460 is full, then a "root array overflow" is said to occur. When too many roots are to be stored in a given root array 460, the roots are no longer tracked. A root array overflow generally occurs when the volume of pointers to root arrays 460 in pool 464 is excessive with respect to the root arrays 460 available for allocation in pool 464. As a result, when performing garbage collection, the pointers in with the card associated with the root array overflow are then typically searched to identify roots. When a root array overflow occurs, 27-bit pointer 478 is set to null. That is, the twenty-seven bits in 27-bit pointer 478 are set to "0."

Reserved bit 480 may be used for any number of suitable purposes which may or may not be directly related to garbage collection processes. Suitable purposes include, but are not limited to, using reserved bit 480 to indicate the age of card mark 452a. It should be appreciated that additional reserved bits may also be included in card mark 452a by reducing the number of bits associated with the pointer in card mark 452a, *e.g.*, 27-bit pointer 478.

Card marks are used in conjunction with a garbage collection process to track cards within a car which include inter-generational pointers to objects in a new generation. In one embodiment, during the execution of a computer program, the card marks may be updated to enable the card marks to remain substantially current. Then, during a garbage collection, the card marks may be checked to identify cards with inter-generational pointers.

整理番号=PA52C757

ページ (24)

When card marks are used to facilitate a garbage collection process during the execution of a computer program, the card marks are generally both searched and updated to identify objects referenced by the card marks. Referring next to Figure 5, the steps associated with executing a program which uses garbage collection will be described in accordance with an embodiment of the present invention. Prior to executing a program, a card array that is associated with the program is initialized in step 502. In one embodiment, a card array, which is associated with a car, includes card marks, or references to card marks, associated with objects used by a program, as mentioned above with respect to Figure 4b. As such, a card array is generally used to track objects used in the program. After the card array is initialized, the program is executed in step 503 until it is desired either to allocate an object or to store a pointer for an object. Allocating an object generally involves assigning memory to an object, whereas storing a pointer for an object involves storing a pointer to another object within a first object.

During execution, there will periodically be instructions to store a pointer. When a pointer is to be stored, as represented by step 504, in step 512, an entry in the card array which corresponds to the object whose pointer is to be stored is identified. Once the card array entry is identified, the card array entry is marked as dirty in step 514. Marking the card array entry, *i.e.*, card mark, as dirty involves setting the dirty flag of the entry, *i.e.*, lowest bit 468 of entry 452a as was described above with respect to Figure 4b. After the card mark is marked as dirty, the pointer is stored in step 516 using any suitable method. Once the pointer is written, process flow returns to step 503 where the program continues to execute. When the program terminates, any re-execution of the program will begin again at step 502 where an array of card marks is initialized.

Execution also generally provides instructions to allocate an object. In one embodiment, returning to step 504, when an object is to be allocated, a determination is made in step 506 as to whether enough new generation memory is

整理番号=PA52C757

ページ (25)

in existence for the object to be allocated. If it is determined in step 506 that there is enough new generation memory for use in allocating an object, then an object is allocated in step 508, and process flow returns to step 503 where a determination is made regarding whether there is a pointer to be stored.

If the determination in step 506 is that there is insufficient new generation memory for an object to be allocated, then process flow proceeds to step 510 where garbage collection is performed. During garbage collection, new generation memory, and possibly old generation memory, may be cleaned to release sections of new generation memory such that an object may be allocated. Although it should be appreciated that any suitable garbage collection method may be used, one particularly suitable garbage collection method will be described below with respect to Figure 6. After garbage collection is performed in step 510, process flow returns to step 506 where a determination is made regarding whether garbage collection has provided enough new generation memory for an object, *i.e.*, the object whose allocation was deemed necessary in step 503, to be allocated.

With reference to Figure 6, the steps associated with performing garbage collection will be described in accordance with an embodiment of the present invention. That is, step 510 of Figure 5 will be described. Garbage collection may be performed in both new generation memory and old generation memory to "free up" enough memory space to enable an object to be allocated. If the system on which garbage collection is to be performed is a multi-threaded system, then the threads are synchronized in step 602 using any suitable method. Once any threads which require synchronization are synchronized, garbage collection is performed within the new generation memory in step 604. The steps associated with one method of performing such a new generation garbage collection will be described below with reference to Figure 7a.

整理番号=PA52C757

ページ (26)

After the new generation garbage collection is performed in step 604, process flow proceeds to step 606 where a determination is made regarding whether it is necessary to perform garbage collection for old generation memory. In the described embodiment, old generation garbage collection is performed periodically. That is, after predetermined amounts of time have elapsed, old generation garbage collection is performed. In other embodiments, however, different or additional criteria may be used in the determination of whether old generation garbage collection is needed.

If it is determined that old generation garbage collection is needed, then old generation garbage collection using a card marking scheme is performed in step 608. Although methods associated with garbage collection performed on old generation memory using a card marking scheme may be widely varied, one suitable old generation garbage collection method will be described below with respect to Figures 8a and 8b.

Once old generation garbage collection is completed in step 608, the overall process of performing garbage collection to enable an object to be allocated is completed. Therefore, process flow proceeds to step 506 of Figure 5, which is the determination of whether enough new generation memory is available for an object to be allocated. Similarly, if the determination in step 606 is that no old generation garbage collection is needed, process flow also proceeds to step 506 of Figure 5.

Figure 7a is a process flow diagram which illustrates the steps associated with a new generation garbage collection, *i.e.*, step 604 of Figure 6, in accordance with an embodiment of the present invention. In step 701, fixed roots are followed. As previously mentioned, fixed roots are roots, *e.g.*, live objects on a stack, which directly point into the new generation memory or the old generation memory which is being cleaned through garbage collection. Following fixed roots typically entails transitively following references of the roots. The actual steps associated with

整理番号=PA52C757

ページ (27)

following roots, fixed or otherwise, will be described below with respect to Figures 7b-7e.

After the fixed roots are followed, the card array is scanned in step 702 to locate the next entry in the card array with either or both a dirty flag or a new generation flag that is set. As previously described, in one embodiment, "setting" a bit refers to assigning a value of "0" the bit, whereas "clearing" a bit refers to assigning a value of "1" the bit. In step 703, a determination is made regarding whether an entry, or a card mark, with either or both a new generation flag or a dirty flag that is set has been found. If such an entry has been found, then process flow proceeds to step 704 where it is determined whether the entry is dirty.

If the determination in step 704 is that the entry is dirty, then the card associated with the entry is marked as clean in step 707. It should be appreciated that marking a card as clean implies marking the card mark associated with the card as clean. After the card is marked as clean in step 707, process flow proceeds to step 708 where all objects in the card are scanned to locate roots which are then followed. As previously mentioned, the steps associated with following the roots will be discussed with reference to Figures 7b-7e below. Once the roots are followed, process flow returns to step 702 where the card array is scanned to locate the next entry, *i.e.*, card mark, with either or both a dirty flag or a new generation flag that is set.

If the determination in step 704 is that the entry is not dirty, then the entry has a new generation flag that is set. In other words, the entry includes an inter-generational pointer. If the entry is not dirty, process flow moves from step 704 to step 706 where a determination is made regarding whether the root array associated with the entry has overflowed. An overflowed root array is a root array which does not have include enough space to accept the insertion of an additional root. If it is determined that the root array associated with the entry is overflowed,

整理番号=PA52C757

ページ (28)

then process flow moves to step 707 where the card associated with the entry is marked as clean.

From step 714, process flow moves to step 718 in which the block associated with the entry is scanned, and the roots associated with the block are followed. Then, process flow returns to step 702, which is the step of scanning the card array to look for the next entry in the card array with the lowest bit set to zero.

When a suitable entry, that is, an entry with the lowest bit set to zero, is not found in step 708, then the roots within the new generation memory are followed until all copied objects have been scanned in step 710. Once all copied objects have been scanned, the new generation garbage collection process is completed.

In general, garbage collection involves following roots, as previously mentioned. Referring next to Figure 7b, the steps associated with following a single root will be described in accordance with an embodiment of the present invention. The process begins with a determination in step 720 of whether the root points into a memory area in which garbage collection is occurring. If it is determined that the root does not point into an area that is being garbage collected, then the process of following the root is completed. On the other hand, if the determination in step 720 is that the root does indeed point into an area in which garbage collection is occurring, then, in step 721, a determination is made as to whether the header field of the card mark holds a forwarding pointer. That is, a determination is made regarding whether a new memory location is identified by a forwarding pointer associated with the card mark.

If the determination is that the header field does hold a forwarding pointer, then process flow moves from step 721 to step 722 where the root is updated to the new location identified by the forwarding counter. After the root is

整理番号 = P A 5 2 C 7 5 7

ページ (29)

updated, the card mark is updated accordingly in step 728. The steps associated with updating a card mark will be described below with reference to Figure 7c. Once the card mark is updated, then the process of following a root is completed.

Returning to step 721, if it is determined that the header field does not hold a forwarding pointer, then the object identified by the root is copied to a new location, and a forwarding pointer is inserted in the header of the object in step 724. The forwarding pointer identifies the new location of the object. When the forwarding pointer has been inserted in the header, the root is updated to the new location in step 726. Then, in step 728, the associated card marking is updated, and the process of following a root is completed.

With respect to Figures 7c-7e, the steps associated with updating a card mark, *i.e.*, step 728 of Figure 7b, will be described in accordance with an embodiment of the present invention. In step 740, a determination is made as to whether the pointer points from the old generation to the new generation. If the pointer does point from the old generation to the new generation, then in step 746, it is determined whether the new generation flag is set to indicate that there is a pointer from the old generation to the new generation.

In step 752, a determination is made regarding whether the root array has overflowed when it is determined in step 746 that the new generation flag is set. If the root array is overflowed, then the process of updating a card mark is completed. If, however, the root array is not overflowed, then a determination is made in step 754 as to whether the root array is full. If it is determined that the root array is not full, process flow proceeds to step 750 in which the root, *i.e.*, the root that is being followed, is inserted into the root array, and the process of updating a card mark is completed. If the root array is determined to be full in step 754, then process flow moves to step 755 where the card mark is set to indicate that

整理番号=PA52C757

ページ (30)

there is an overflow of the root array. After the card mark is set, the card mark update process is completed.

Returning to step 746 and the determination of whether the new generation flag is set, when it is determined that the new generation pointer is not set, then in step 747, the new generation pointer is set to indicate that the pointer points from the old generation to the new generation. A root array is allocated and the card mark is set to point to the allocated root array in step 748. From step 748, process flow proceeds to step 749 which is the determination of whether the allocation of the root array was successful. If the allocation of the root array was successful, then the followed root is inserted in the root array in step 750. On the other hand, if the allocation of the root array was not successful, then process flow proceeds to step 755 where the card mark is set to indicate that the root array is overflowed. In one embodiment, setting the card mark to indicate that the root array is overflowed involves setting the pointer, *i.e.*, the 27-bit pointer, in the card mark to null.

If the determination in step 740 was that the pointer does not point from the old generation to the new generation, then process flow proceeds to step 742 which is the determination of whether there is a pointer which points to a lower train in the old generation. If it is determined that the pointer points to a lower train in the old generation, then a determination is made in step 760 as to whether the new generation flag is set to indicate that there is a pointer from the old generation to the new generation. If the indication is that the new generation flag is set, then process flow proceeds to step 775, which is the determination of whether the lower train flag is set. In the described embodiment, the lower train flag is the third lowest bit in the card mark, and indicates whether the pointer points into a lower train, as described above. If the determination is that the lower train flag is set, the process of updating the card mark is completed. Alternatively, if the determination is that the lower train flag is not set, then the lower train flag is set

整理番号=PA52C757

ページ (31)

in step 776. Once the lower train overflow flag is set, then the card mark update process is completed.

If it is determined in step 760 that the new generation flag is not set, then process flow moves to step 766 where it is determined whether the lower train flag is set. If the determination in step 766 is that the lower train flag is not set, then the lower train flag is set in step 767 to indicate the existence of a pointer to a lower train. Once the lower train flag is set, a root array is allocated and the card mark is set to point to the allocated root array in step 768. A determination is then made in step 769 regarding whether the allocation of the new root array was successful. If the allocation was successful, then in step 770, the followed root is inserted in the root array, in the event that the root is not already present in the root array. Then, the process of updating the card mark is completed. On the other hand, if the allocation of a new root array in step 769 was not successful, then process flow moves from step 769 to step 762 where the card mark is set to indicate a root array overflow. After the card mark is set to indicate a root array overflow, the process of updating the card mark is completed.

Returning to step 766, if the determination is that the lower train flag is set to indicate that there is a pointer to a lower train, then a determination is made in step 772 regarding whether the corresponding root array is overflowed. If the root array is overflowed, then the card mark is set to indicate a root array overflow in step 762. Alternatively, if the root array is determined not to be overflowed in step 772, then the followed root is inserted in the root array in step 770, if the root is not already present in the root array.

Referring back to step 742, which is the step of determining if a pointer points to a lower train in the old generation, if it is determined that a pointer does not point to a lower train in the old generation, then in step 744, a determination is made as to whether a pointer points to a lower car in the same train. If a pointer

整理番号=PA52C757

ページ (32)

does not point to a lower car in the same train, then the card mark update process is completed. If, however, the determination is that a pointer does point to a lower car in the same train, then process flow proceeds to step 780, which is the determination of whether the new generation flag is set to indicate that there is a pointer to a new generation.

If the indication in step 780 is that the new generation flag is set, then a determination is made in step 782 as to whether the same train flag is set. That is, a determination is made regarding whether there is a pointer into a lower car of the same train. If the same train flag is set, then the process of updating a card mark is completed. Alternatively, if the same train flag is not set, then the same train flag is set in step 783. Once the same train flag is set, then the process of updating a card mark is completed.

When it is determined in step 780 that the new generation flag is not set, then process flow proceeds to step 786 where a determination is made regarding whether the lower train flag of the card mark is set to indicate that there is a pointer to a lower train. If the lower train flag is set, then process flow proceeds to step 782 where a determination is made regarding whether the same train flag is set.

If the lower train flag is not set, or indicates that pointers to a lower train are not in existence, then process flow moves from step 786 to step 788 where a determination is made regarding whether the same train flag of the card mark is set. Step 788, in other words, is the determination of whether the card mark indicates that there is a pointer to a lower card of the same train. If it is determined that the same train flag is set, then in step 790, it is determined whether a root array associated with the card mark is overflowed. If the root array is overflowed, then the card mark is set to indicate that the root array is overflowed in step 793. Alternatively, if it is determined that the root array is not overflowed,

整理番号=PA52C757

ページ (33)

then, in step 792, the root is inserted in the root array in the event that the root is not already present in the root array. After the root is inserted in the block array, if necessary, the process of updating a card mark is completed.

Returning to step 788, if it is determined that the same train flag is not set, then a root array is allocated in step 794. Further, the card mark is set to point to the allocated root array. In step 796, it is determined whether the allocation of the root array was successful. If the allocation of the root array was not successful, then the card mark is set to indicate a root array overflow in step 782. On the other hand, if the allocation of the root array was successful, then process flow proceeds to step 792 where the root is inserted into the root array, if the root is not already present in the root array.

Figures 8a and 8b are process flow diagrams which illustrate the steps associated with performing garbage collection in old generation memory, *i.e.*, step 608 of Figure 6, in accordance with an embodiment of the present invention. In step 802, fixed roots or, in this embodiment, roots which directly point into old generation memory, are followed. The steps associated with following the fixed roots, according to an embodiment of the present invention, were described above with respect to Figures 7b-7e.

Roots from the new generation are followed in step 804. That is, the associations of roots which point from the new generation to the old generation are followed. Process flow moves from step 804 to step 806 in which the card array, or the array of card marks, is scanned in order to locate the next entry which has a lower train flag that is set. In step 808, it is determined whether such an entry, *i.e.*, an entry which has a lower train flag that is set, has been found. If it is determined that such an entry has been found, then a determination is made in step 812 regarding whether the entry that was found has either, or both, a new generation flag that is set or a root array which is overflowed. If either, or both, the new

整理番号=PA52C757

ページ (34)

generation flag is set or a root array is overflowed, then in step 814,, all the objects in the card are scanned for roots, and the roots are followed. Again, the steps associated with following a root were previously discussed with respect to Figures 7b-7e. After the roots are followed, process flow returns to step 806 where the card array is scanned to locate the next entry which has a lower train flag that is set.

If it is determined in step 812 that the entry, *i.e.*, card mark, that was found in step 808 does not have a new generation flag that is set or a root array that is overflowed, then process flow moves to step 816 where the root associated with the entry is scanned, and the roots associated with the root array are followed. After the roots are followed, the card array is scanned in step 806 for the next entry which has a lower train flag that is set.

Returning to step 808, if scanning the card array looking for an entry which has a lower train flag that is set results in no entry being found, then in step 820, the roots are followed within the old generation until all copied objects have been scanned. In one embodiment, objects in one train which are referenced from other trains are copied into those trains. When all copied objects have been scanned, a determination is made in step 830 regarding whether a root which points into the lowest train was encountered during the scanning process. If the determination is that a root into the lowest train was not encountered, then the entire lowest train is released in step 840. In other words, old generation memory held by the lowest train is freed. The entire lowest train may be freed due to the fact that if no pointers into the lowest train exist, the lowest train contains only garbage. It should be appreciated that cyclic structures which reference each other and are located in different cars of the same train may be collected when the entire lowest train is freed. Once the entire lowest train is freed, the steps associated with an old generation garbage collection are completed.

整理番号=PA52C757

ページ (35)

If it is determined in step 830 that a root into the lowest train was encountered, then in step 832, the card array is scanned to look for the next entry, *i.e.*, card mark, which has a same train flag that is set. In step 834, a determination is made regarding whether an entry with a same train flag that is set has been found. If the determination is that an entry with a same train flag that is set has not been found, then the roots are followed within the old generation until all copied objects have been scanned in step 836. Scanning all copied objects involves, in one embodiment, a transitive search for pointers. Once all copied objects have been scanned, the lowest car in the lowest train is released in step 837, and an old generation garbage collection process is completed.

Returning to step 834, if the determination is that an entry, or card mark, with a same train flag that is set has been found, then process flow proceeds to step 838 where a determination is made regarding whether the entry has a new generation flag that is set, a lower train flag that is set, or a root array that is overflowed. If the entry has any of or all of a new generation flag that is set, then in step 850, all objects in the card are scanned for roots, which are followed. Once the roots are followed, process flow returns to step 832 where the card array is once again scanned to look for the next entry which has a same train flag that is set.

If the determination in step 838 is that the entry found in step 834 does not include a new generation flag that is set, a lower train flag that is set, or a root array that is overflowed, then the root array associated with the entry is scanned in step 856, and the roots in the root array are followed. After the roots are followed, process flow returns to step 832 where the card array is scanned to look for the next entry which has a same train flag that is set.

It should be appreciated that a computer program which uses inter-generational garbage collection in accordance with the present invention may be implemented on a variety of different computer systems. Figure 9 illustrates a

整理番号=PA52C757

ページ (36)

typical, general purpose computer system suitable for implementing the present invention. The computer system 930 includes any number of processors 932 (also referred to as central processing units, or CPUs) that are coupled to memory devices including primary storage devices 934 (typically a read only memory, or ROM) and primary storage devices 936 (typically a random access memory, or RAM). As is well known in the art, ROM acts to transfer data and instructions uni-directionally to the CPU 932, while RAM is used typically to transfer data and instructions in a bi-directional manner. Both primary storage devices 934, 936 may include any suitable computer-readable media. A secondary storage medium 938, which is typically a mass memory device, is also coupled bi-directionally to CPU 932 and provides additional data storage capacity. The mass memory device 938 is a computer-readable medium that may be used to store programs including computer code, data and the like and is typically a storage medium such as a hard disk or a tape that are generally slower than primary storage devices 934, 936. Mass memory storage device 938 may take the form of a magnetic or paper tape reader or some other well-known device. It will be appreciated that the information retained within the mass memory device 938, may, in appropriate cases, be incorporated in standard fashion as part of RAM 936 as virtual memory. A specific primary storage device 934 such as a CD-ROM may also pass data uni-directionally to the CPU.

CPU 932 is also coupled to one or more input/output devices 940 that may include, but are not limited to, devices such as video monitors, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-known input devices such as, of course, other computers. Finally, CPU 932 optionally may be coupled to a computer or telecommunications network, *e.g.*, an Internet network or an intranet network, using a network connection as shown generally at 912. With such a network connection, it is contemplated that the CPU 932 might receive information from the network, or might output information to the network in the course of performing the above-described method

整理番号=PA52C757

ページ (37)

steps. Such information, which is often represented as a sequence of instructions to be executed using CPU 932, may be received from and outputted to the network in the form of a computer data signal embodied in a carrier wave. The above-described devices and materials will be familiar to those of skill in the computer hardware and software arts.

Although only a few embodiments of the present invention have been described, it should be understood that the present invention may be embodied in many other specific forms without departing from the spirit or scope of the invention. By way of example, steps involved with new generation garbage collection and the old generation garbage collection may be reordered. Steps may also be removed or added without departing from the spirit or the scope of the present invention.

While generational garbage collection has been described as being performed in a new generation and an old generation of memory, generational garbage collection may be performed over many generations of memory. That is, generational garbage collection methods of the present invention may be performed on memory that is divided into multiple generations, *e.g.*, a new generation, an "intermediate generation," and an old generation, which are segregated by the age of objects within each generation. It should be appreciated that for embodiments in which there are multiple generations, an inter-generational pointer may generally point between any two generations.

The card mark has been described as being a 32-bit word which includes a 27-bit pointer, a reserved bit, a same train flag, a lower train flag, a new generation flag, and a dirty flag. However, it should be appreciated that the card mark may be varied without departing from the spirit of the scope of the present invention. For example, the card mark may include fewer than 32 bits or more than 32 bits, depending at least in part upon the particular requirements of a given system.

整理番号=PA52C757

ページ (38)

Alternatively, the bits within a card mark may also be widely varied. By way of example, in one embodiment, the 32-bits within a card mark may include a 26-bit pointer and two "status indicator" bits, or tag bits. The status indicator bits may be arranged such that different combinations of values for the status indicator bits indicate whether a card mark is dirty, clean, includes a new generation pointer, or does not include a new generation pointer. The card mark may also include a lower train flag which indicates whether there are any pointers to a lower train, a same train flag which indicates whether there are any pointers within a given train to a lower car in that train, a lower train overflow flag, and a same train overflow flag.

Further, although the lowest bits within a card mark are described as being flags which are used to indicate, for example, whether the card mark is dirty, as described above, it should be appreciated that the flags within a card mark may be located in any suitable position within the card mark. The bits which represent flags may be the highest bits within a card mark. Alternatively, the bits may be interspersed throughout a card mark.

Although the size of a car in old generation memory has been described as being fixed, it should be appreciated that, in one embodiment, the size of a car may be dynamically allocated. That is, the size of a car may be determined based upon the particular requirements of objects which are to be placed within the car. By way of example, the size of a car may be allocated during the course of garbage collection process to minimize the number of pointers which originate with objects within the car without departing from the spirit or the scope of the present invention.

It should be appreciated that conventional methods for garbage collection may occasionally be implemented within old generation memory. By way of example, in some cases, it may be possible for old generation memory to become full. When old generation memory is full, performing old generation garbage

整理番号=PA52C757

ページ (39)

collection on a car may not free up a sufficient amount of memory for immediate use. As such, a conventional garbage collection process, such as a mark sweep collection process, may be implemented to free as much old generation memory as possible in the event that the old generation memory is full without departing from the spirit or the scope of the present invention. Therefore, the present examples are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

4 Brief Description of Drawings

Figure 1 is a diagrammatic representation of an area of computer memory which contains objects in accordance with prior art.

Figure 2 is a diagrammatic representation of an interface between a root and memory which includes a new generation and an old generation in accordance with prior art.

Figure 3 is a diagrammatic representation of a memory which is associated with a remembered set in accordance with prior art.

Figure 4a is a diagrammatic representation of an old generation memory which is segmented into trains in accordance with an embodiment of the present invention.

Figure 4b is a diagrammatic representation of an array of card marks and an associated pool of root arrays in accordance with an embodiment of the present invention.

Figure 5 is a process flow diagram which illustrates the steps associated with executing a computer program in accordance with an embodiment of the present invention.

Figure 6 is a process flow diagram which illustrates a process of performing a garbage collection, i.e., step 510 of Figure 5, in accordance with an embodiment of the present invention.

整理番号= P A 5 2 C 7 5 7

ページ (40)

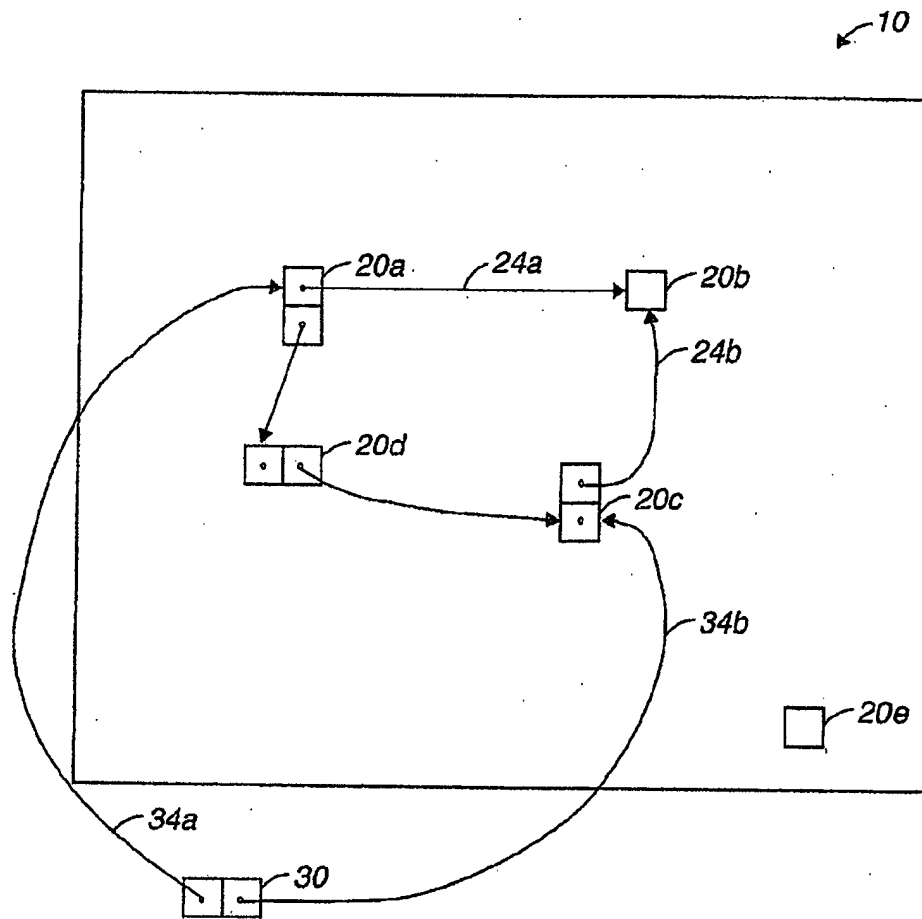
Figure 7a is a process flow diagram which illustrates the steps associated with a garbage collection in new generation memory, *i.e.*, step 604 of Figure 6, in accordance with an embodiment of the present invention.

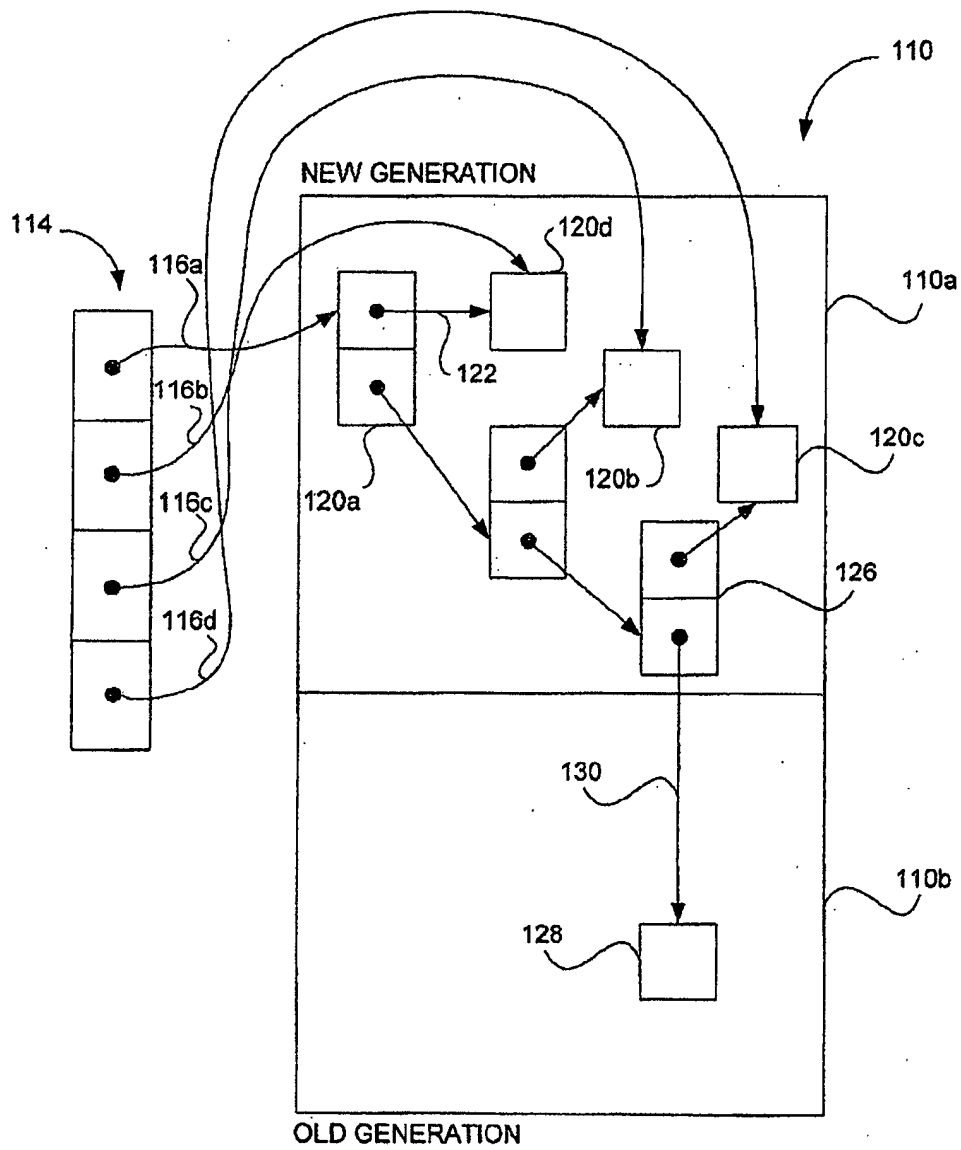
Figure 7b is a process flow diagram which illustrates the steps associated with following a single root in accordance with an embodiment of the present invention.

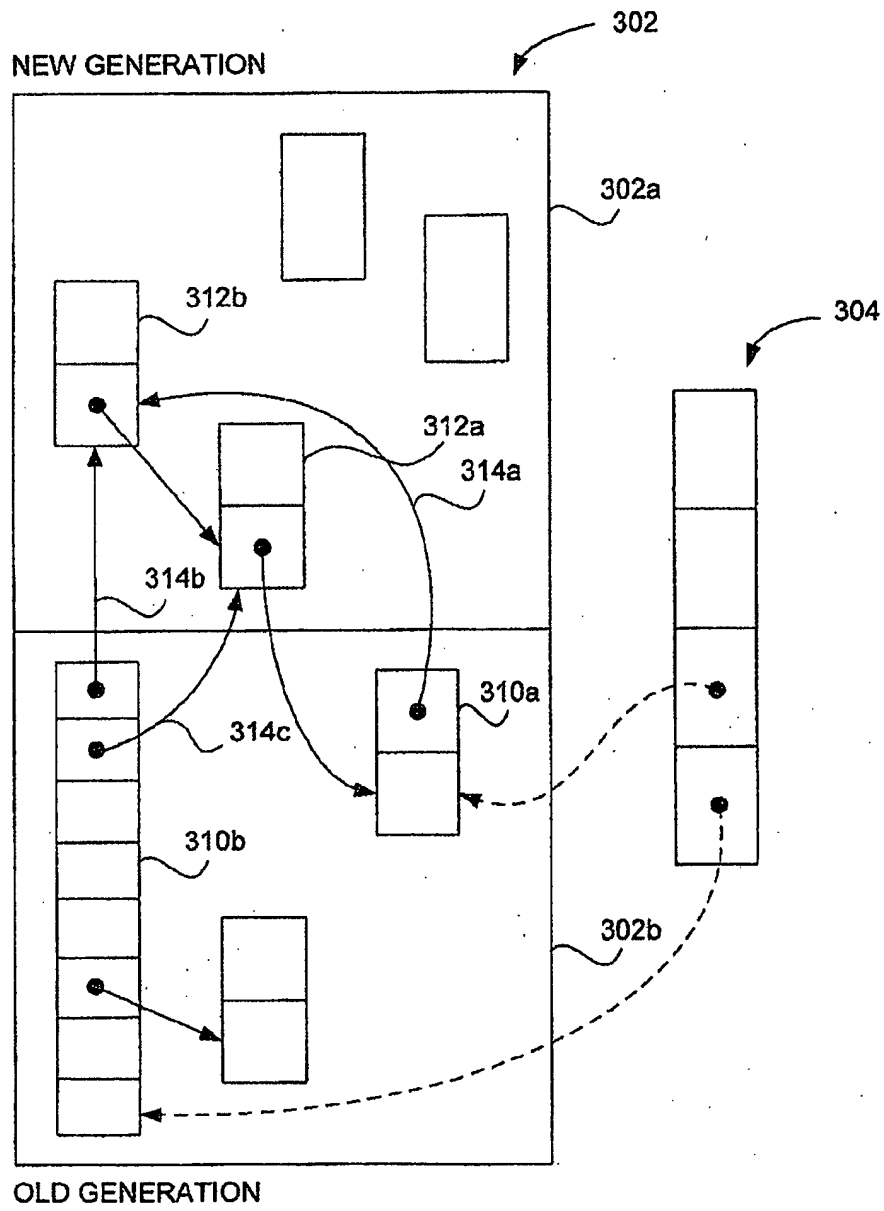
Figures 7c-7e are a process flow diagram which illustrates the steps associated with updating a card mark, *i.e.*, step 728 of Figure 7b, in accordance with an embodiment of the present invention.

Figures 8a and 8b are a process flow diagram which illustrates the steps associated with performing an old generation garbage collection, *i.e.*, step 608 of Figure 6, in accordance with an embodiment of the present invention.

Figure 9 is a diagrammatic representation of a computer system suitable for implementing the present invention.

*Figure 1*

*Figure 2*

*Figure 3*

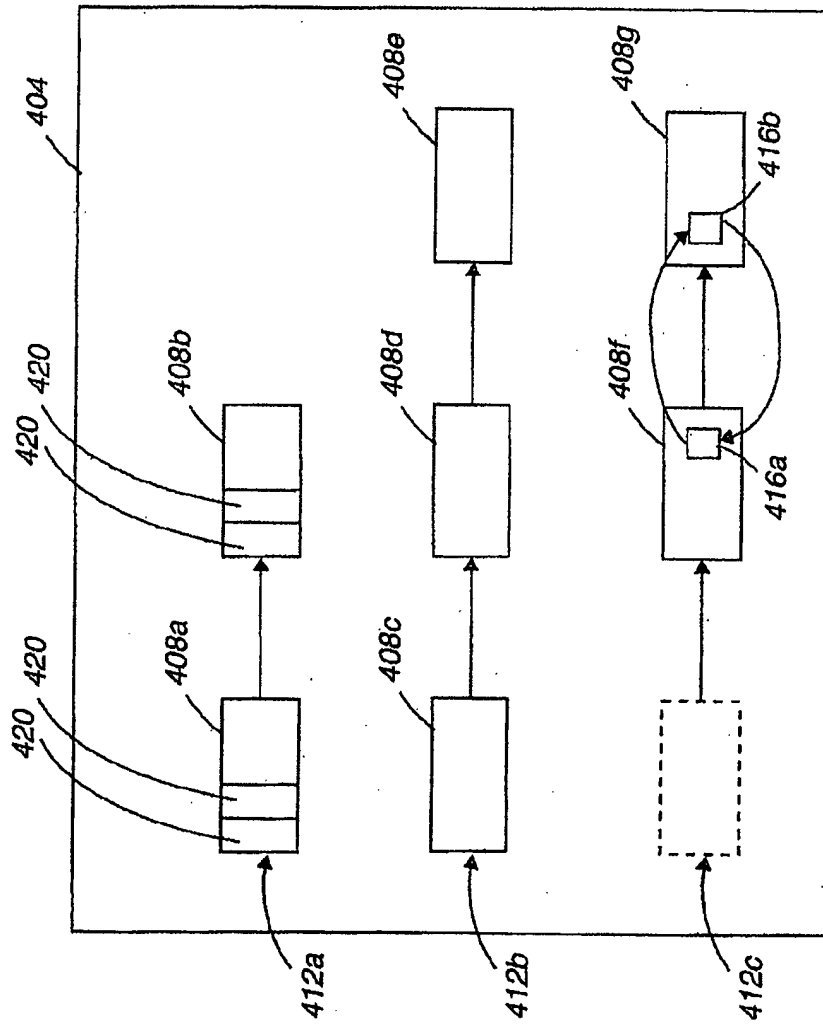


Figure 4a

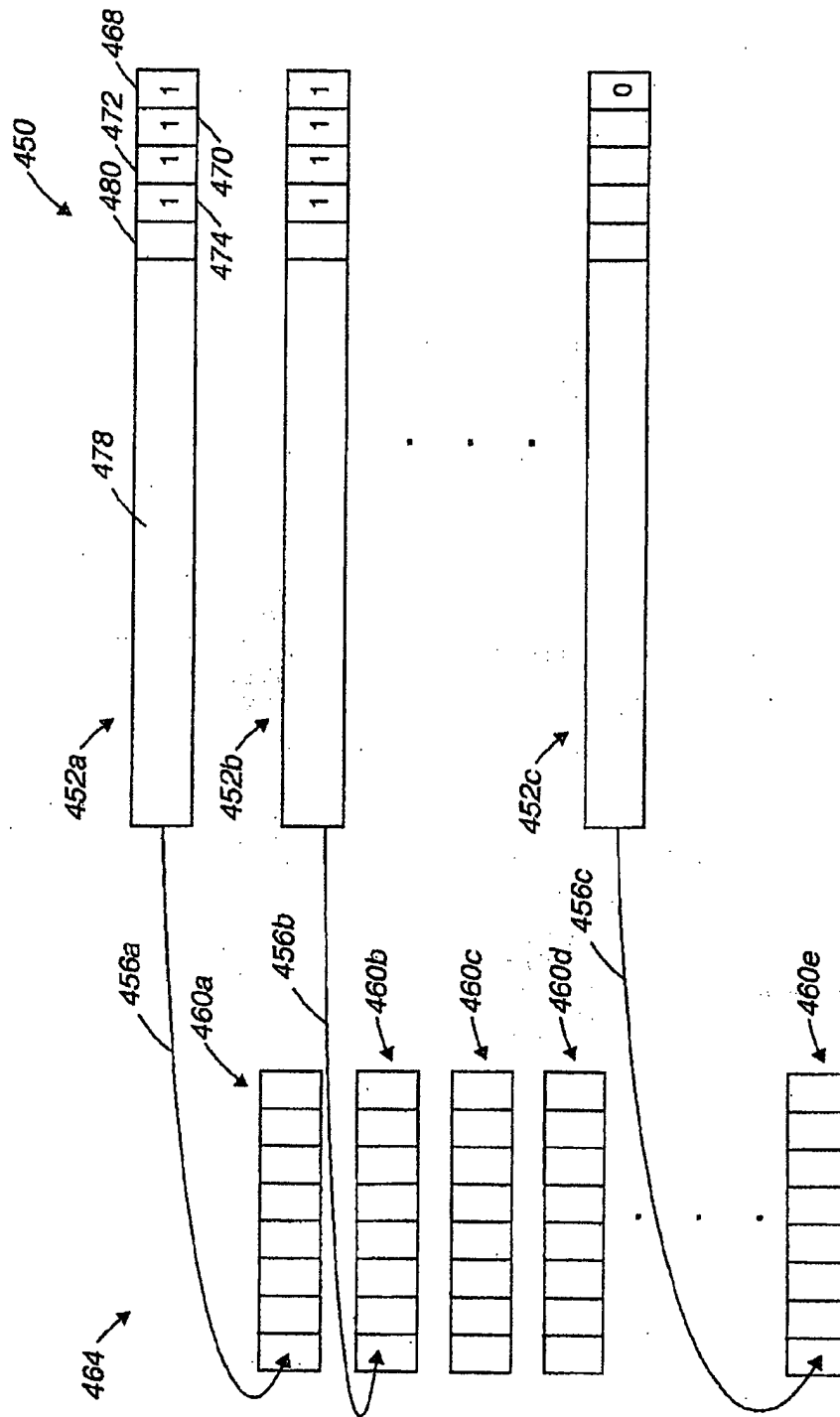
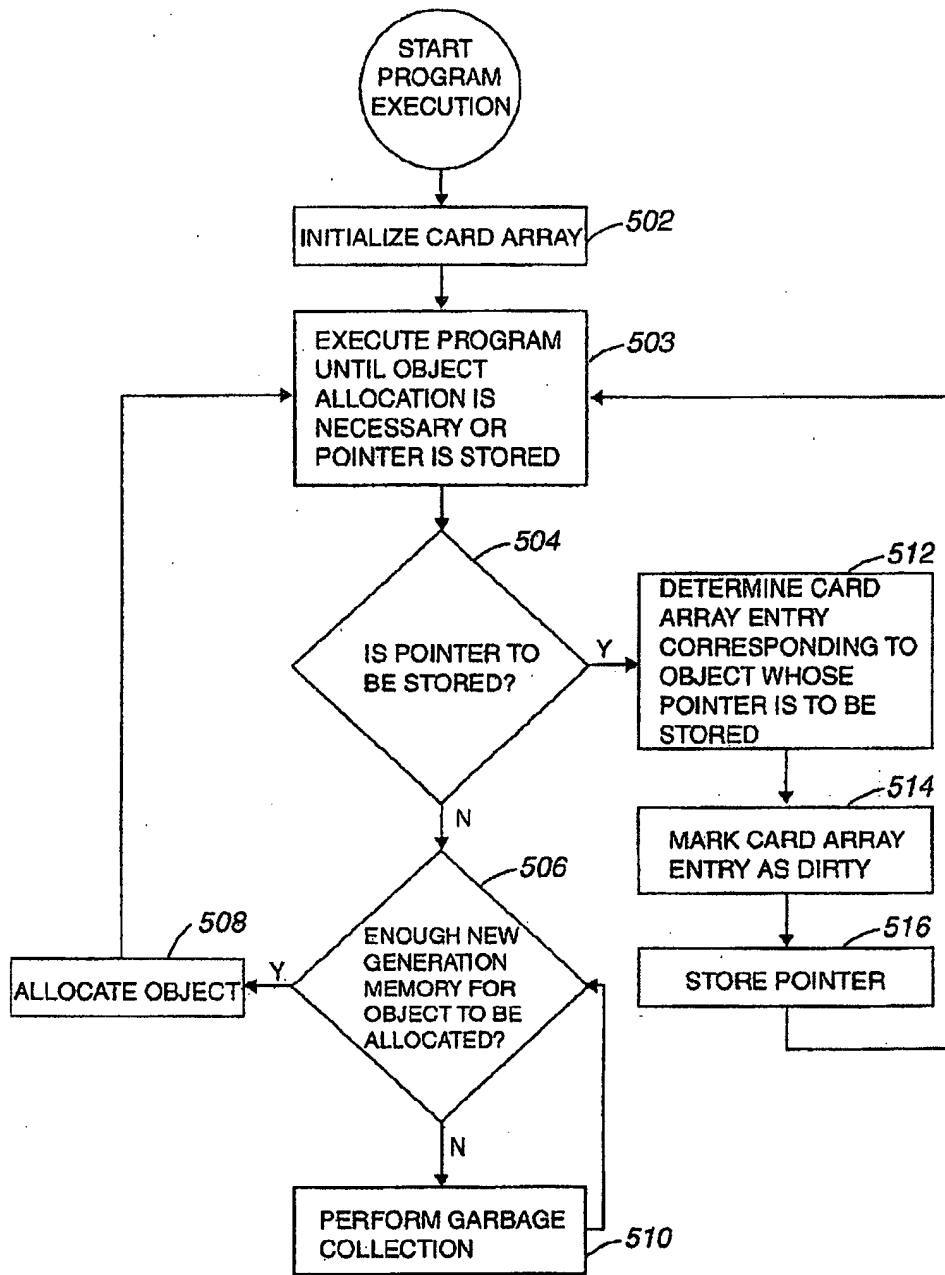
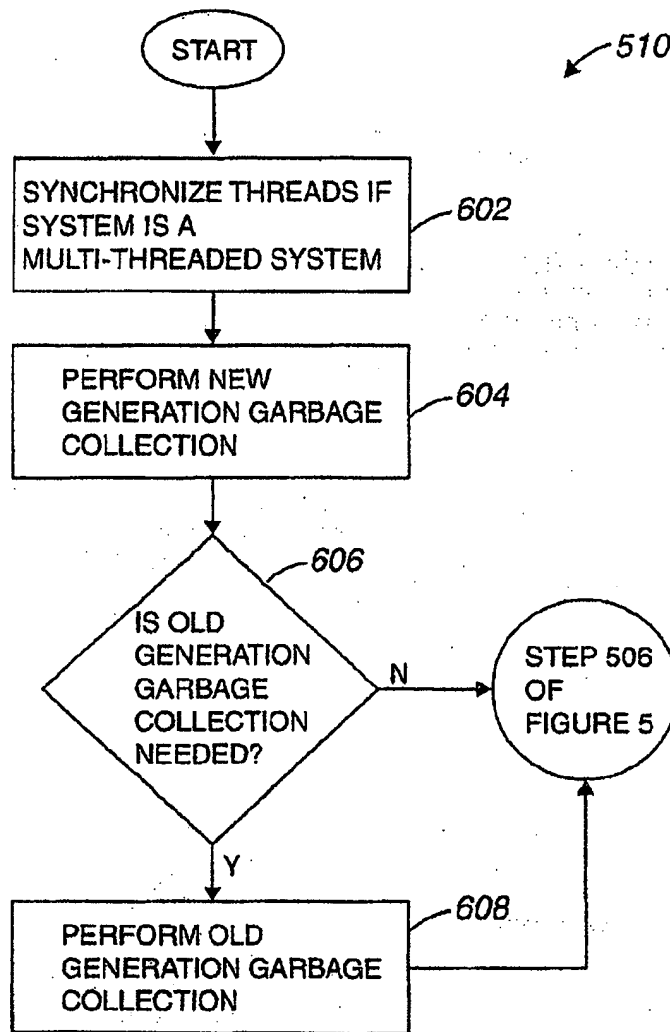
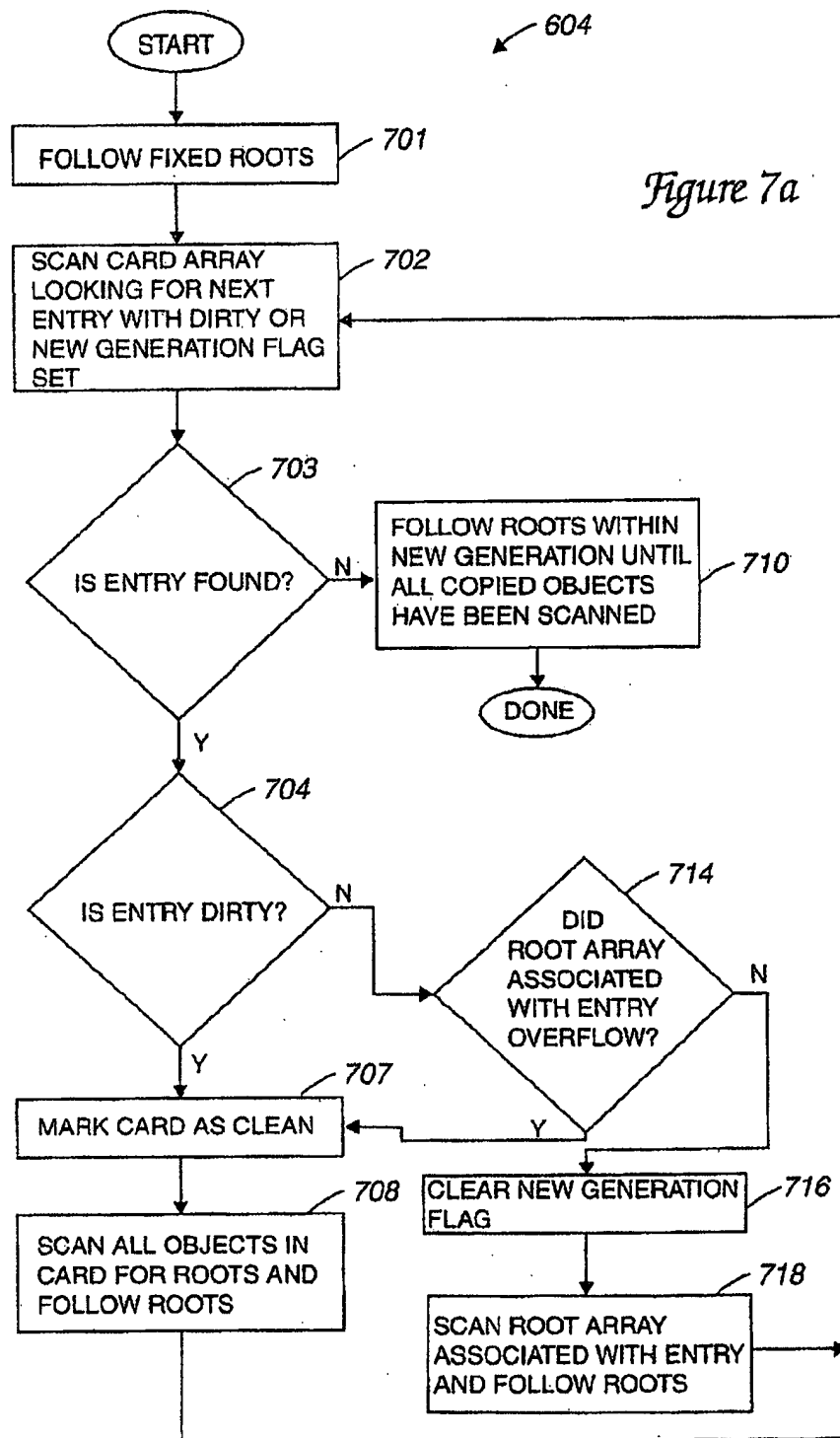


Figure 46

*Figure 5*

*Figure 6*



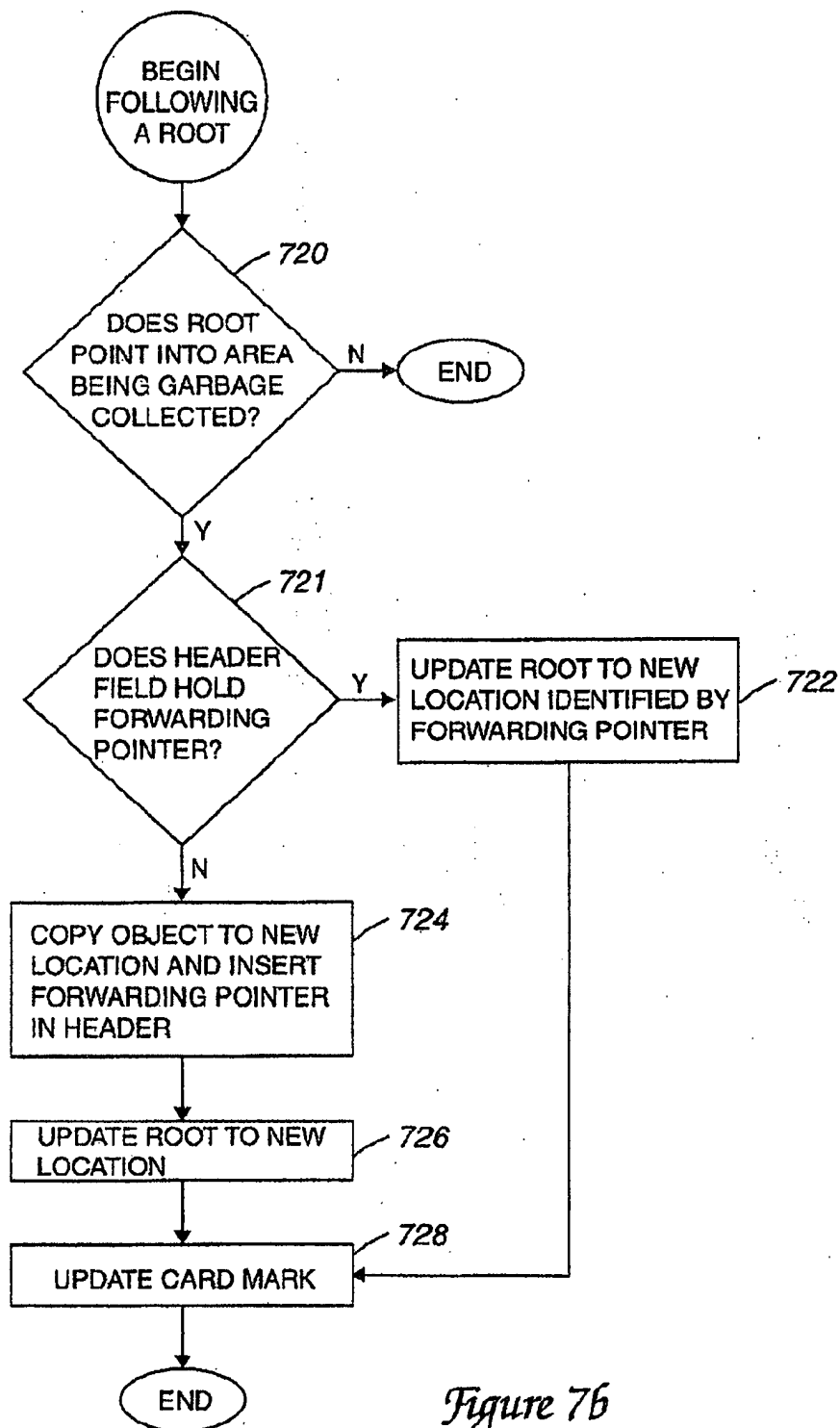


Figure 76

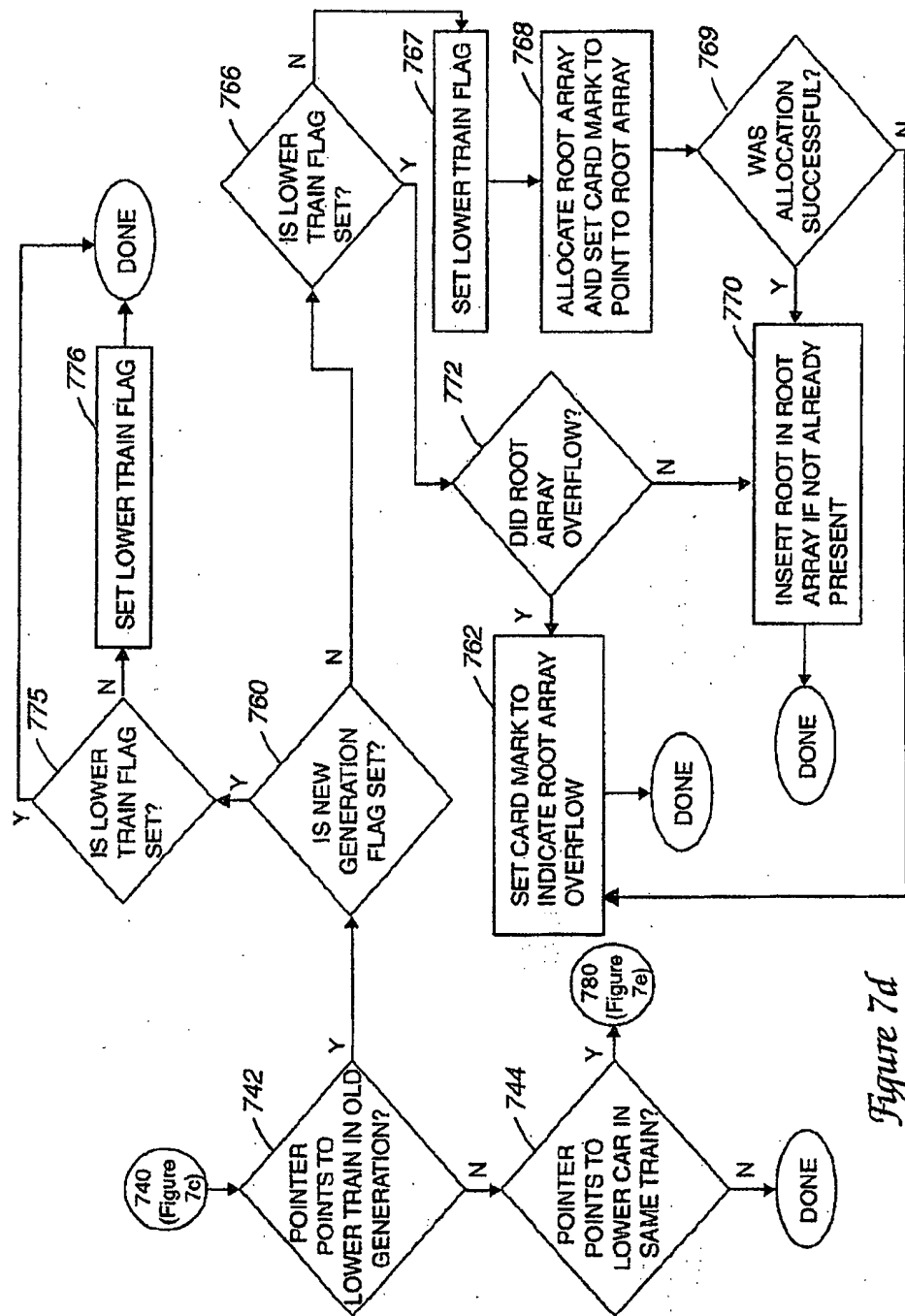
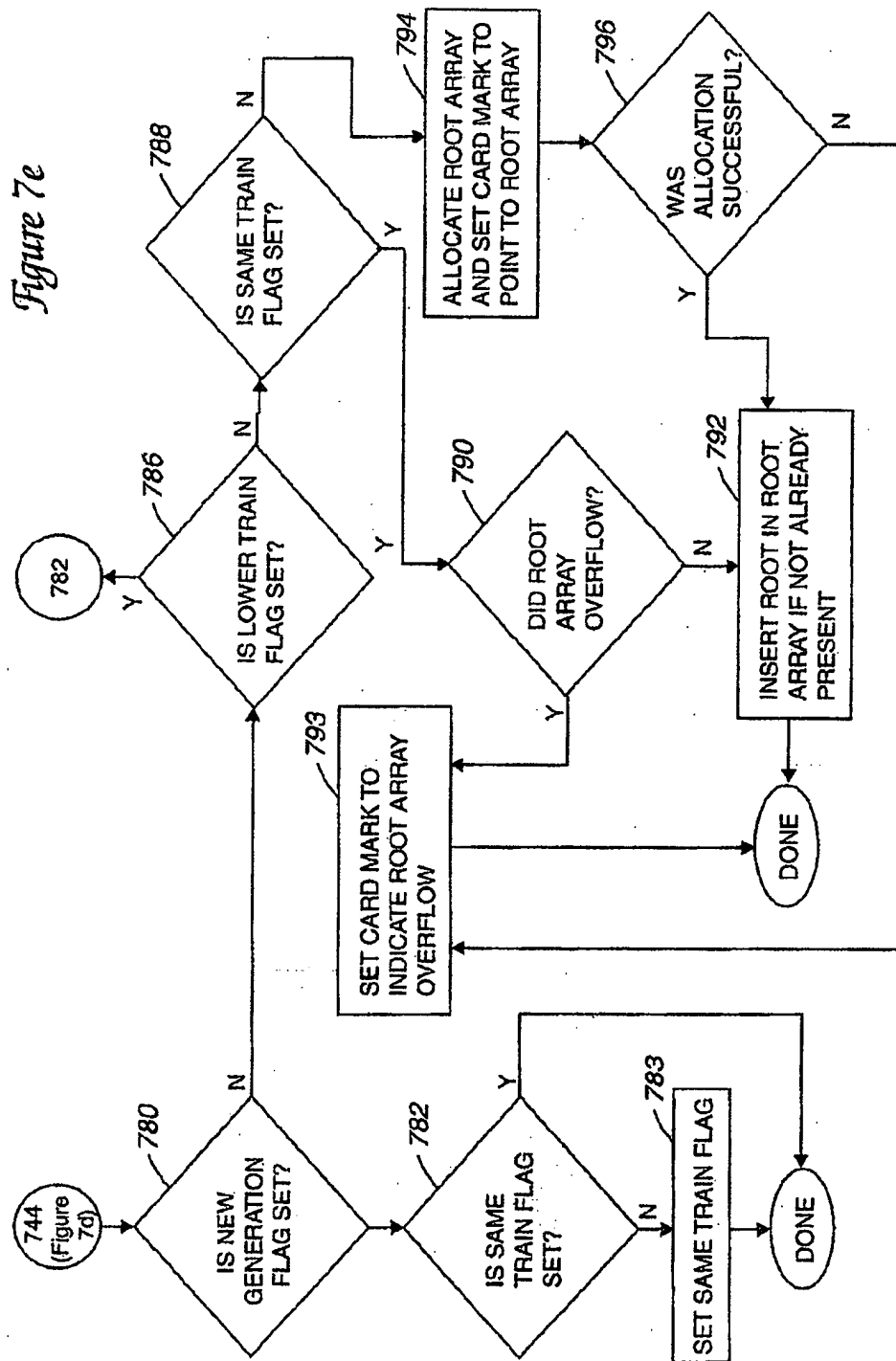
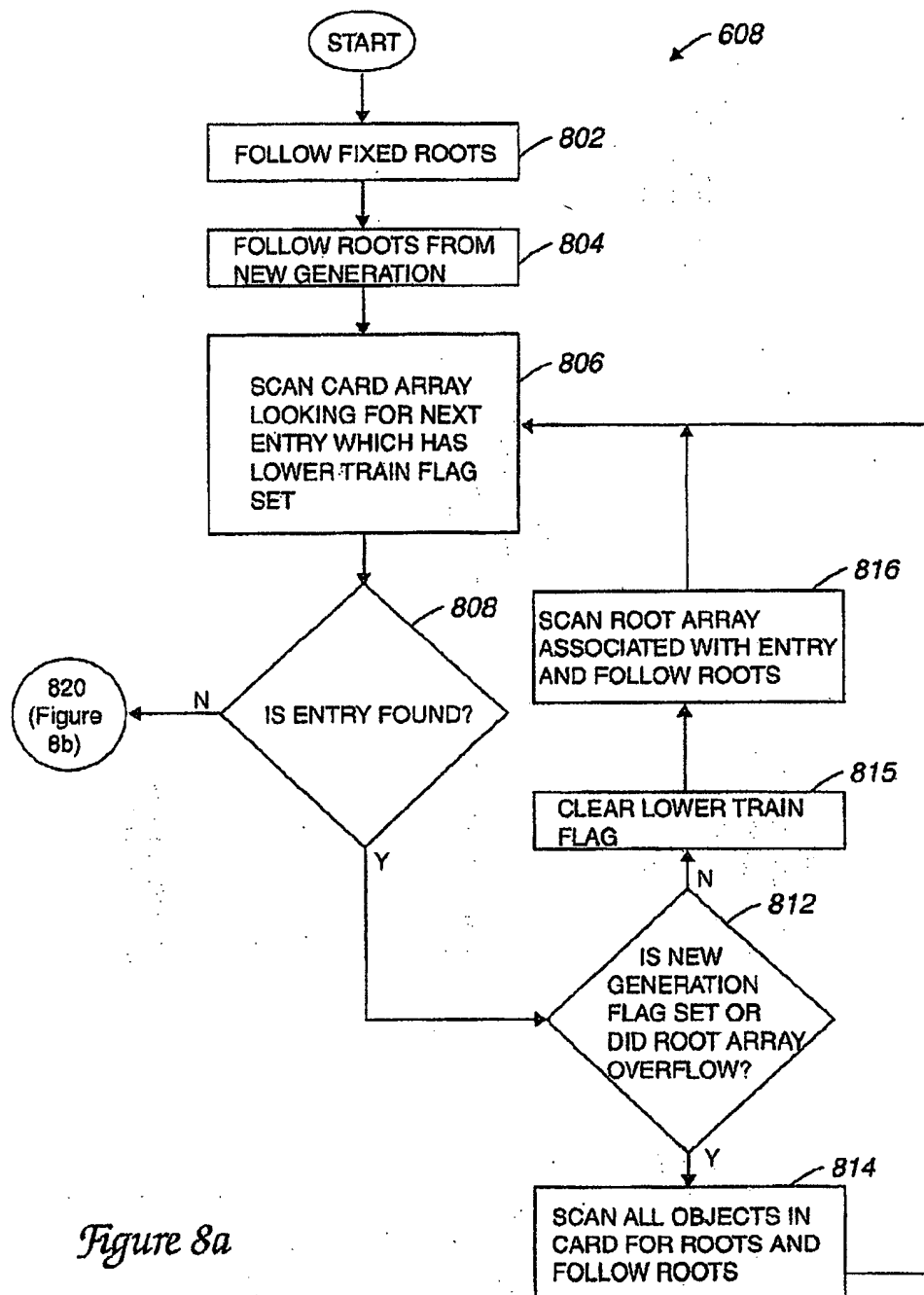


Figure 7d





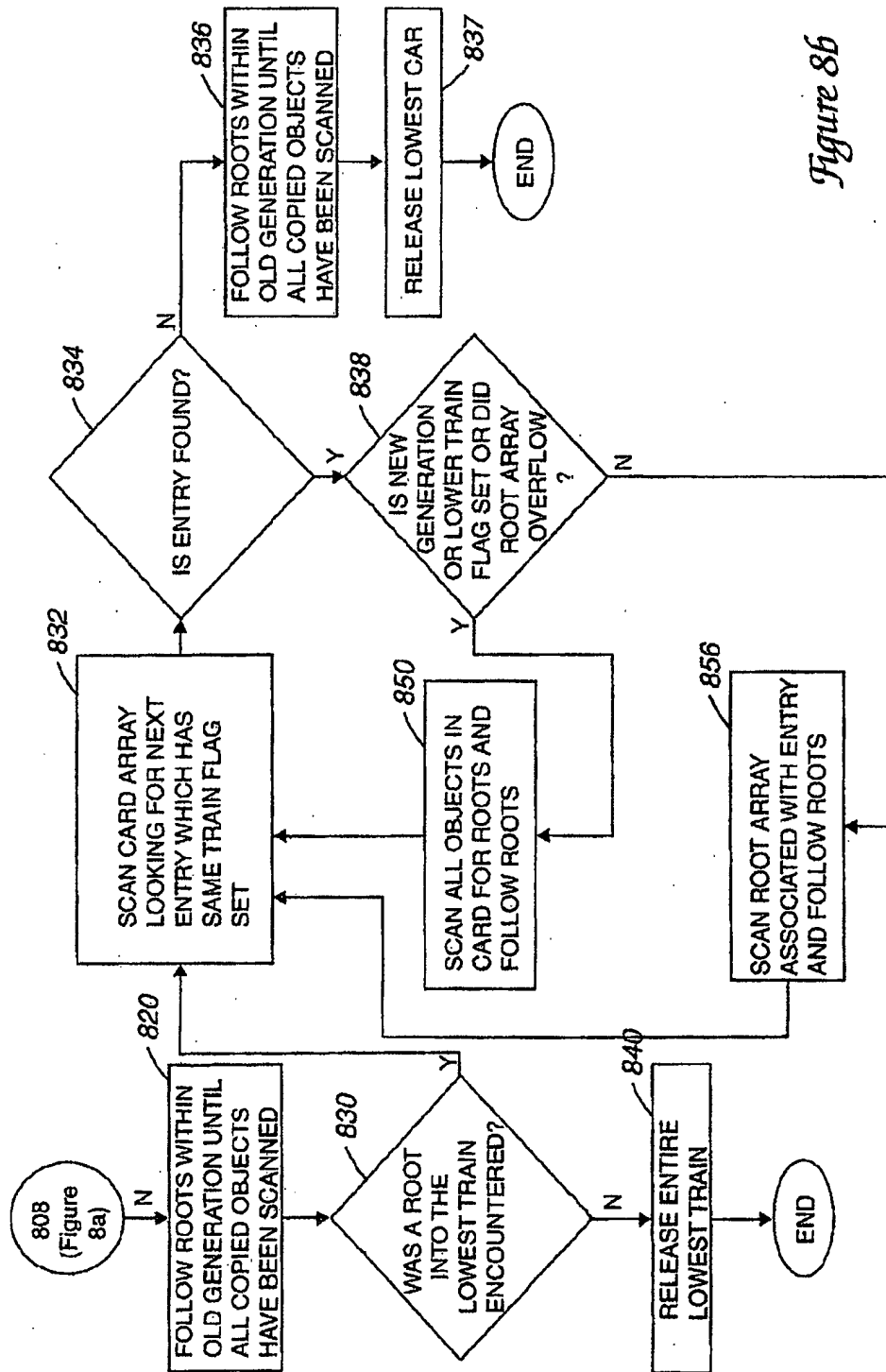
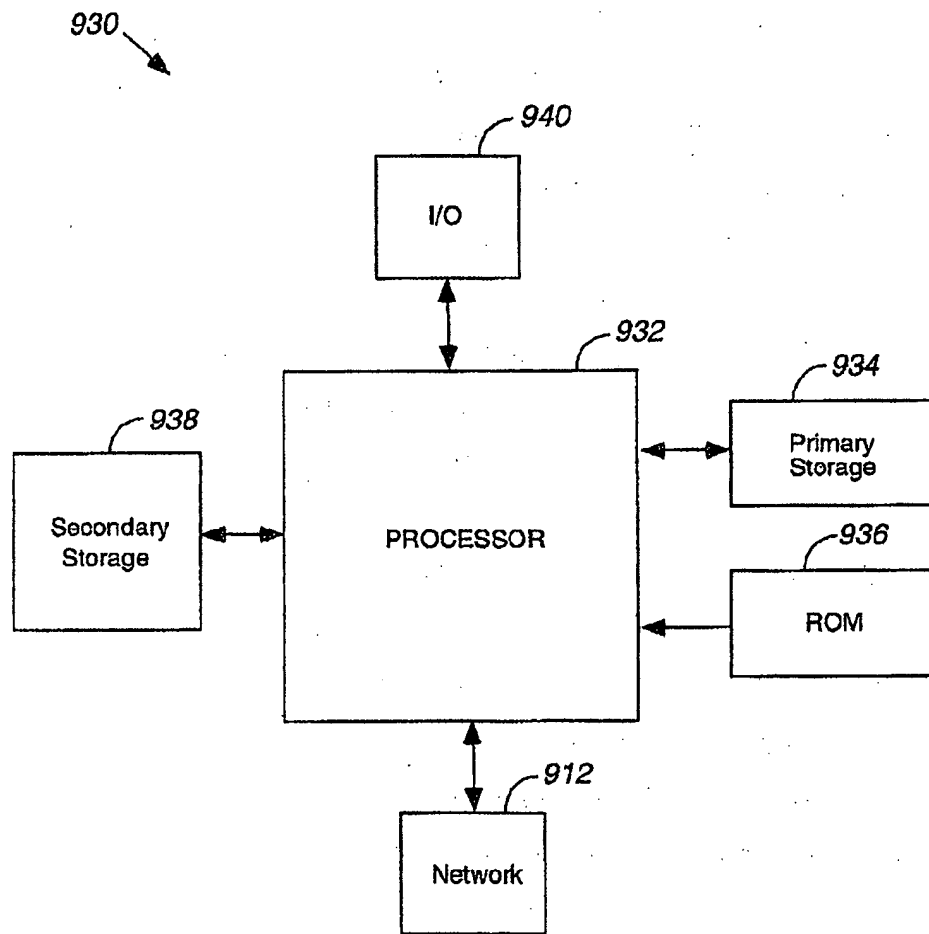


Figure 8b

*Figure 9*

1 Abstract

The present invention relates to methods and apparatus for performing generational garbage collection within computer memory. According to one aspect of the present invention, a computer-implemented method for dynamically managing memory which includes a first memory section and a second memory section that is divided into a plurality of blocks each having an associated marker, includes performing a first garbage collection on the first memory section. The method also includes performing a second garbage collection on a selected one of the blocks in the second memory section. A third garbage collection is performed on the selected block in the second memory section. The third garbage collection includes determining whether the selected block includes a first object which references a second object which is not included in the selected block based at least in part on a status indicated by the marker associated with the selected block. The status includes an indication of whether the reference to the second object was stored after the second garbage collection was performed, and if the reference to the second object was stored after the second garbage collection was performed, a new root array is created using the selected marker.

2 Representative Drawing

Fig. 5